IBM DB2 10.1
for Linux, UNIX, and Windows

Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611

Updated February, 2014

IBM

SC27-4541-01

IBM DB2 10.1
for Linux, UNIX, and Windows

Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611

Updated February, 2014

SC27-4541-01

Note

Before using this information and the product it supports, read the general information under Appendix B, “Notices,” on page 1051.

Edition Notice

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

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

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

Contents

Chapter 20. Time Travel Query using temporal tables 247

Chapter 21. Constraints 299

Chapter 22. Indexes 319

##

##

#

##

##

##

About this book

This book provides information from the DB2® for Linux, UNIX, and Windows documentation to cover all the objectives that are described in the DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611.

- Part 1, "DB2 Server management," on page 1 provides information about how to configure and manage DB2 servers, instances, and databases, how to use autonomic features, and how to schedule jobs and use visual explain with IBM® Data Studio.
- Part 2, "Client-to-server communications," on page 75 provides information about clients, types of clients, how to configure communication protocols and clients, how to establish database connections, and how to use LDAP for authentication.
- Part 3, "Physical design and business rules implementation," on page 107 provides information about defining database objects such as tables and views, and implementing business rules by using table constraints, views, and triggers. Also, it provides information about new capabilities for tables such as temporal tables and the multi-temperature storage.
- Part 4, "Monitoring DB2 Activity," on page 409 provides information about tasks that are associated with examining the operational status of your database, interfaces for database and workload monitoring, and tools for obtaining information about access plans and troubleshooting problems.
- Part 5, "DB2 commands for database administration," on page 543 provides information about DB2 commands for performing administration tasks such as moving data, organizing data, collecting catalog statistics, and binding applications.
- Part 6, "High availability," on page 721 provides information about data integrity actions, how to back up databases and table spaces, how to use HADR and its new capabilities, and high availability characteristics in DB2 pureScale® environments.
- Part 7, "Security," on page 943 provides information about the DB2 security model, authorization, authorities, privileges, roles, trusted contexts, label-based access control (LBAC), row and column access control (RCAC), and the DB2 audit facility.

Passing the DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611 is one of the requirements to obtain the *IBM Certified Database Administrator - DB2 10.1 for Linux, UNIX, and Windows* certification. For complete details about this certification and its requirements, visit http://www.ibm.com/certify/certs/08002107.shtml.

Who should use this book

This book is for database administrators and other DB2 database users with intermediate to advanced administration skills who want to prepare for the certification Exam 611. For complete details about the exam, visit http://www.ibm.com/certify/tests/ovr611.shtml.

Part 1. DB2 Server management

DB2 Server management provides information about how to configure and manage DB2 servers, instances, and databases, how to use autonomic features, and how to schedule jobs and use visual explain with IBM Data Studio.

A data server refers to a computer where the DB2 database engine is installed. The DB2 engine is a full-function, robust database management system that includes optimized SQL support based on actual database usage and tools to help manage the data.

IBM offers a number data server products, including data server clients that can access all the various data servers. For a complete list of DB2 data server products, features available, and detailed descriptions and specifications, visit the product page at the following URL: http://www.ibm.com/software/data/db2/linux-unix-windows/.

Chapter 1. Instances

An *instance* is a logical database manager environment where you catalog databases and set configuration parameters. Depending on your needs, you can create more than one instance on the same physical server providing a unique database server environment for each instance.

Note: For non-root installations on Linux and UNIX operating systems, a single instance is created during the installation of your DB2 product. Additional instances cannot be created.

You can use multiple instances to do the following:

- Use one instance for a development environment and another instance for a production environment.
- Tune an instance for a particular environment.
- Restrict access to sensitive information.
- Control the assignment of SYSADM, SYSCTRL, and SYSMAINT authority for each instance.
- Optimize the database manager configuration for each instance.
- Limit the impact of an instance failure. In the event of an instance failure, only one instance is affected. Other instances can continue to function normally.

Multiple instances will require:

- Additional system resources (virtual memory and disk space) for each instance.
- More administration because of the additional instances to manage.

The instance directory stores all information that pertains to a database instance. You cannot change the location of the instance directory once it is created. The directory contains:

- The database manager configuration file
- The system database directory
- The node directory
- The node configuration file (`db2nodes.cfg`)
- Any other files that contain debugging information, such as the exception or register dump or the call stack for the DB2 database processes.

Terminology:

Bit-width
The number of bits used to address virtual memory: 32-bit and 64-bit are the most common. This term might be used to refer to the bit-width of an instance, application code, external routine code. 32-bit application means the same things as 32-bit width application.

32-bit DB2 instance
A DB2 instance that contains all 32-bit binaries including 32-bit shared libraries and executables.

64-bit DB2 instance
A DB2 instance that contains 64-bit shared libraries and executables, and

also all 32-bit client application libraries (included for both client and server), and 32-bit external routine support (included only on a server instance).

Designing instances

DB2 databases are created within DB2 instances on the database server. The creation of multiple instances on the same physical server provides a unique database server environment for each instance.

For example, you can maintain a test environment and a production environment on the same computer, or you can create an instance for each application and then fine-tune each instance specifically for the application it will service, or, to protect sensitive data, you can have your payroll database stored in its own instance so that owners of other instances (on the same server) cannot see payroll data.

The installation process creates a default DB2 instance, which is defined by the DB2INSTANCE environment variable. This is the instance that is used for most operations. However, instances can be created (or dropped) after installation.

When determining and designing the instances for your environment, note that each instance controls access to one or more databases. Every database within an instance is assigned a unique name, has its own set of system catalog tables (which are used to keep track of objects that are created within the database), and has its own configuration file. Each database also has its own set of grantable authorities and privileges that govern how users interact with the data and database objects stored in it. Figure 1 shows the hierarchical relationship among systems, instances, and databases.

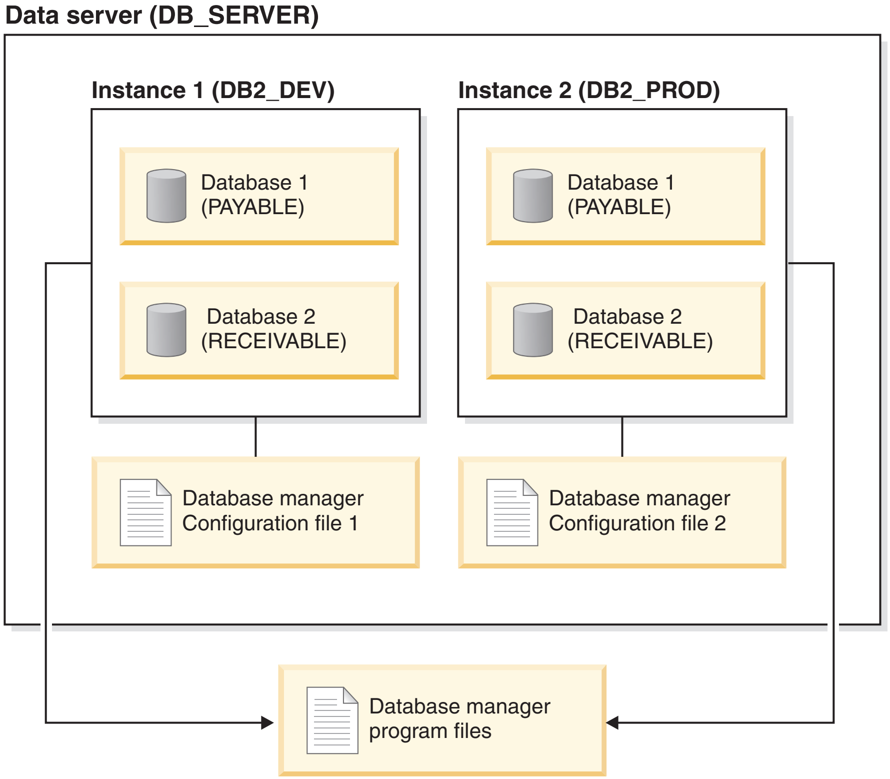


Figure 1. Hierarchical relationship among DB2 systems, instances, and databases

You also must be aware of another particular type of instance called the *DB2 administration server* (DAS). The DAS is a special DB2 administration control point used to assist with the administration tasks only on other DB2 servers. A DAS must be running if you want to use the Client Configuration Assistant to discover the remote databases or the graphical tools that come with the DB2 product, for example, the IBM Data Studio. There is only one DAS in a DB2 database server, even when there are multiple instances.

Important: The DB2 Administration Server (DAS) has been deprecated in Version 9.7 and might be removed in a future release. The DAS is not supported in DB2 pureScale environments. Use software programs that use the Secure Shell protocol for remote administration. For more information, see " DB2 administration server (DAS) has been deprecated" at .

Once your instances are created, you can attach to any other instance available (including instances on other systems). Once attached, you can perform maintenance and utility tasks that can only be done at the instance level, for example, create a database, force applications off a database, monitor database activity, or change the contents of the database manager configuration file that is associated with that particular instance.

Default instance

As part of your DB2 installation procedure, you can create an initial instance of the database manager. The default name is `DB2_01` in Version 9.5 or later releases.

On Linux and UNIX, the initial instance can be called anything you want within the naming rules guidelines. The instance name is used to set up the directory structure.

To support the immediate use of this instance, the following registry variables are set during installation:

- The environment variable **DB2INSTANCE** is set to `DB2_01`.
- The registry variable **DB2INSTDEF** is set to `DB2_01`.

These settings establish "DB2" as the default instance. You can change the instance that is used by default, but first you have to create an additional instance.

Before using the database manager, the database environment for each user must be updated so that it can access an instance and run the DB2 database programs. This applies to all users (including administrative users).

On Linux and UNIX operating systems, sample script files are provided to help you set the database environment. The files are: `db2profile` for Bourne or Korn shell, and `db2cshrc` for C shell. These scripts are located in the `sqllib` subdirectory under the home directory of the instance owner. The instance owner or any user belonging to the instance's SYSADM group can customize the script for all users of an instance. Use `sqllib/userprofile` and `sqllib/usercshrc` to customize a script for each user.

The blank files `sqllib/userprofile` and `sqllib/usercshrc` are created during instance creation to allow you to add your own instance environment settings. The `db2profile` and `db2cshrc` files are overwritten during an instance update in a DB2 fix pack installation. If you do not want the new environment settings in the `db2profile` or `db2cshrc` scripts, you can override them using the corresponding user script, which is called at the end of the `db2profile` or `db2cshrc` script. During an instance upgrade (using the **db2iupgrade** command), the user scripts are copied over so that your environment modifications will still be in use.

The sample script contains statements to:

- Update a user's **PATH** by adding the following directories to the existing search path: the bin, adm, and misc subdirectories under the sqllib subdirectory of the instance owner's home directory.
- Set the **DB2INSTANCE** environment variable to the instance name.

Instance directory

The instance directory stores all information that pertains to a database instance. The location of the instance directory cannot be changed after it is created.

The instance directory contains:

- The database manager configuration file
- The system database directory
- The node directory
- The node configuration file (db2nodes.cfg)
- Other files that contain debugging information, such as the exception or register dump or the call stack for the DB2 processes.

On Linux and UNIX operating systems, the instance directory is located in the *INSTHOME*/sqllib directory, where *INSTHOME* is the home directory of the instance owner. The default instance can be called anything you want within the naming rules guidelines.

On Windows operating systems, the instance directory is located under the /sqllib directory where the DB2 database product was installed. The instance name is the same as the name of the service, so it should not conflict. No instance name should be the same as another service name. You must have the correct authorization to create a service.

In a partitioned database environment, the instance directory is shared between all database partition servers belonging to the instance. Therefore, the instance directory must be created on a network share drive that all computers in the instance can access.

db2nodes.cfg

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

Multiple instances (Linux, UNIX)

It is possible to have more than one instance on a Linux or UNIX operating system if the DB2 product was installed with root privileges. Although each instance runs simultaneously, each is independent. Therefore, you can only work within one instance of the database manager at a time.

Note: To prevent environmental conflicts between two or more instances, you should ensure that each instance has its own home directory. Errors will be returned when the home directory is shared. Each home directory can be in the same or a different file system.

The instance owner and the group that is the System Administration (SYSADM) group are associated with every instance. The instance owner and the SYSADM group are assigned during the process of creating the instance. One user ID or username can be used for only one instance, and that user ID or username is also referred to as the *instance owner*.

Each instance owner must have a unique home directory. All of the configuration files necessary to run the instance are created in the home directory of the instance owner's user ID or username. If it becomes necessary to remove the instance owner's user ID or username from the system, you could potentially lose files associated with the instance and lose access to data stored in this instance. For this reason, you should dedicate an instance owner user ID or username to be used exclusively to run the database manager.

The primary group of the instance owner is also important. This primary group automatically becomes the system administration group for the instance and gains SYSADM authority over the instance. Other user IDs or usernames that are members of the primary group of the instance owner also gain this level of authority. For this reason, you might want to assign the instance owner's user ID or username to a primary group that is reserved for the administration of instances. (Also, ensure that you assign a primary group to the instance owner user ID or username; otherwise, the system-default primary group is used.)

If you already have a group that you want to make the system administration group for the instance, you can assign this group as the primary group when you create the instance owner user ID or username. To give other users administration authority on the instance, add them to the group that is assigned as the system administration group.

To separate SYSADM authority between instances, ensure that each instance owner user ID or username uses a different primary group. However, if you choose to have a common SYSADM authority over multiple instances, you can use the same primary group for multiple instances.

Multiple instances (Windows)

It is possible to run multiple instances of the DB2 database manager on the same computer. Each instance of the database manager maintains its own databases and has its own database manager configuration parameters.

Note: The instances can also belong to different DB2 copies on a computer that can be at different levels of the database manager. If you are running a 64-bit Windows system, you can install 32-bit DB2, or 64-bit DB2 but they cannot co-exist on the same machine.

An instance of the database manager consists of the following:

- A Windows service that represents the instance. The name of the service is same as the instance name. The display name of the service (from the Services panel) is the instance name, prefixed with the “DB2 - ” string. For example, for an instance named “DB2”, there exists a Windows service called “DB2” with a display name of “DB2 - *DB2 Copy Name* - DB2”.

Note: A Windows service is not created for client instances.

- An instance directory. This directory contains the database manager configuration files, the system database directory, the node directory, the Database Connection Services (DCS) directory, all the diagnostic log and dump files that are associated with the instance. The instance directory varies from edition to edition of the Windows family of operating systems; to verify the default directory on Windows, check the setting of the **DB2INSTPROF** environment variable using the command **db2set DB2INSTPROF**. You can also change the default instance directory by changing the **DB2INSTPROF** environment variable. For example, to set it to `c:\DB2PROFS`:
  - Set **DB2INSTPROF** to `c:\DB2PROFS` using the **db2set.exe -g** command
  - Run **DB2ICRT.exe** command to create the instance.
- When you create an instance on Windows operating systems, the default locations for user data files, such as instance directories and the `db2cli.ini` file, are the following directories:
  - On the Windows XP and Windows 2003 operating systems: `Documents and Settings\All Users\Application Data\IBM\DB2\`*`Copy Name`*
  - On the Windows 2008 and Windows Vista (and later) operating system: `Program Data\IBM\DB2\`*`Copy Name`*

  where *Copy Name* represents the DB2 copy name.

  Note: The location of the `db2cli.ini` file might change based on whether the Microsoft ODBC Driver Manager is used, the type of data source names (DSN) used, the type of client or driver being installed, and whether the registry variable **DB2CLIINIPATH** is set.

Creating instances

Although an instance is created as part of the installation of the database manager, your business needs might require you to create additional instances.

Before you begin

If you belong to the Administrative group on Windows, or you have root user authority on Linux or UNIX operating systems, you can add additional instances. The computer where you add the instance becomes the instance-owning computer (node zero). Ensure that you add instances on a computer where a DB2 administration server resides. Instance IDs should not be root or have password expired.

Restrictions

- On Linux and UNIX operating systems, additional instances cannot be created for non-root installations.
- If existing user IDs are used to create DB2 instances, make sure that the user IDs:
  - Are not locked
  - Do not have expired passwords

Procedure

To add an instance using the command line:

Enter the command: db2icrt *instance_name*.
When creating instance on an AIX® server, you must provide the fenced user id, for example:

```
DB2DIR/instance/db2icrt -u db2fenc1 db2inst1
```

When using the **db2icrt** command to add another DB2 instance, you should provide the login name of the instance owner and optionally specify the authentication type of the instance. The authentication type applies to all databases created under that instance. The authentication type is a statement of where the authenticating of users will take place.
You can change the location of the instance directory from **DB2PATH** using the **DB2INSTPROF** environment variable. You require write-access for the instance directory. If you want the directories created in a path other than **DB2PATH**, you have to set **DB2INSTPROF** before entering the **db2icrt** command.
For DB2 Enterprise Server Edition (ESE), you also must declare that you are adding a new instance that is a partitioned database system. In addition, when working with a ESE instance having more than one database partition, and working with Fast Communication Manager (FCM), you can have multiple connections between database partitions by defining more TCP/IP ports when creating the instance.
For example, for Windows operating systems, use the **db2icrt** command with the **-r** *port_range* parameter. The port range is shown as follows, where the *base_port* is the first port that can be used by FCM, and the *end_port* is the last port in a range of port numbers that can be used by FCM:

```
-r:base_port,end_port
```

Modifying instances

Instances are designed to be as independent as possible from the effects of subsequent installation and removal of products. On Linux and UNIX, you can update instances after the installation or removal of executables or components. On Windows, you run the **db2iupdt** command.

In most cases, existing instances automatically inherit or lose access to the function of the product being installed or removed. However, if certain executables or components are installed or removed, existing instances do not automatically inherit the new system configuration parameters or gain access to all the additional function. The instance must be updated.

If the database manager is updated by installing a Program Temporary Fix (PTF) or a patch, all the existing database instances should be updated using the **db2iupdt** command (root installations) or the **db2nrupdt** command (non-root installations).

You should ensure you understand the instances and database partition servers you have in an instance before attempting to change or delete an instance.

Updating the instance configuration (Linux, UNIX)

To update the configuration for root instances on Linux or UNIX operating systems, use the **db2iupdt** command. To update non-root instances, run the **db2nrupdt** command.

About this task

Running the **db2iupdt** command updates the specified instance by performing the following:

- Replaces the files in the `sqllib` subdirectory under the home directory of the instance owner.
- If the node type has changed, then a new database manager configuration file is created. This is done by merging relevant values from the existing database manager configuration file with the default database manager configuration file for the new node type. If a new database manager configuration file is created, the old file is backed up to the `backup` subdirectory of the `sqllib` subdirectory under the home directory of the instance owner.

The **db2iupdt** command is located in the *DB2DIR*/`instance` directory, where *DB2DIR* is the location where the current version of the DB2 database product is installed.

Restrictions

This task applies to root instances only.

Procedure

To update an instance from the command line, enter:

```
db2iupdt InstName
```

The *InstName* is the login name of the instance owner.

Example

- If you installed DB2 Workgroup Server Edition or DB2 Enterprise Server Edition after the instance was created, enter the following command to update that instance:

  ```
  db2iupdt -u db2fenc1 db2inst1
  ```

- If you installed the DB2 Connect™ Enterprise Edition after creating the instance, you can use the instance name as the Fenced ID also:

  ```
  db2iupdt -u db2inst1 db2inst1
  ```

- To update client instances, invoke the following command:

  ```
  db2iupdt db2inst1
  ```

Updating the instance configuration (Windows)

To update the instance configuration on Windows, use the **db2iupdt** command.

About this task

Running the **db2iupdt** command updates the specified instance by performing the following:

- Replaces the files in the `sqllib` subdirectory under the home directory of the instance owner.
- If the node type is changed, then a new database manager configuration file is created. This is done by merging relevant values from the existing database manager configuration file with the default database manager configuration file for the new node type. If a new database manager configuration file is created, the old file is backed up to the `backup` subdirectory of the `sqllib` subdirectory under the home directory of the instance owner.

The **db2iupdt** command is found in \sqllib\bin directory.

Procedure

To update the instance configuration, issue the **db2iupdt** command. For example:

```
db2iupdt InstName
```

The *InstName* is the login name of the instance owner.
There are other optional parameters associated with this command:

/h: *hostname*
: Overrides the default TCP/IP host name if there are one or more TCP/IP host names for the current computer.

/p: *instance-profile-path*
: Specifies the new instance profile path for the updated instance.

/r: *baseport*,*endport*
: Specifies the range of TCP/IP ports used by the partitioned database instance when running with multiple database partitions.

/u: *username*,*password*
: Specifies the account name and password for the DB2 service.

Auto-starting instances

You can enable instances to start automatically after each system restart. The steps necessary to accomplish this task differ by operating system.

About this task

On Windows operating systems, the database instance that is created during installation is set as auto-started by default.

On Linux, UNIX and Windows operating systems, an instance created by using **db2icrt** is set as a manual start.

Procedure

To configure an instance to start automatically:

- On Windows operating systems, you must go to the Services panel and change the property of the DB2 service there.
- On Linux and UNIX operating systems, perform the following steps:
  1. Verify that the instance's **startAtBoot** global registry field value is set to 1 by checking the output of the following command:

     ```
     db2greg -getinstrec instancename='<instance name>'
     ```

     If the **startAtBoot** global registry field value is not set to 1, set the value to 1 by running the following command:

     ```
     db2greg -updinstrec instancename='<instance name>'!startatboot=1
     ```

  2. Enable the instance to auto-start after each system restart, by running the following command:

     ```
     db2iauto -on <instance name>
     ```

  where *instance_name* is the login name of the instance.

Starting instances (Linux, UNIX)

You might need to start or stop a DB2 database during normal business operations. For example, you must start an instance before you can perform some of the following tasks: connect to a database on the instance, precompile an application, bind a package to a database, or access host databases.

Before you begin

Before you start an instance on your Linux or UNIX operating system:

1. Log in with a user ID or name that has SYSADM, SYSCTRL, or SYSMAINT authority on the instance; or log in as the instance owner.
2. Run the startup script as follows, where *INSTHOME* is the home directory of the instance you want to use:

   ```
   . INSTHOME/sqllib/db2profile      (for Bourne or Korn shell)
   source INSTHOME/sqllib/db2cshrc   (for C shell)
   ```

Procedure

To start the instance:

- From the command line, enter the **db2start** command. The DB2 database manager applies the command to the current instance.
- From IBM Data Studio, open the task assistant for starting the instance. For more information, see IBM Data Studio: Administering databases with task assistants.

Starting instances (Windows)

You might need to start or stop a DB2 instance during normal business operations. For example, you must start an instance before you can perform some of the following tasks: connect to a database on the instance, precompile an application, bind a package to a database, or access a host database.

Before you begin

In order to successfully launch the DB2 database instance as a service, the user account must have the correct privilege as defined by the Windows operating system to start a Windows service. The user account can be a member of the Administrators, Server Operators, or Power Users group. When extended security is enabled, only members of the DB2ADMNS and Administrators groups can start the database by default.

About this task

By default, the **db2start** command launches the DB2 database instance as a Windows service. The DB2 database instance on Windows can still be run as a process by specifying the **/D** parameter on the **db2start** command. The DB2 database instance can also be started as a service by using the Control Panel or the **NET START** command.

When running in a partitioned database environment, each database partition server is started as a Windows service. You cannot use the **/D** parameter to start a DB2 instance as a process in a partitioned database environment.

Procedure

To start the instance:

- From the command line, enter the **db2start** command. The DB2 database manager applies the command to the current instance.
- From IBM Data Studio, open the task assistant for starting the instance.

Attaching to and detaching from instances

On all platforms, to attach to another instance of the database manager, which might be remote, use the **ATTACH** command. To detach from an instance, use the **DETACH** command.

Before you begin

More than one instance must exist.

Procedure

- To attach to an instance:
  - Enter the **ATTACH** command from the command line.
  - Call the sqleatin API from a client application.
- To detach from an instance:
  - Enter the **DETACH** from the command line.
  - Call the sqledtin API from a client application.

Example

For example, to attach to an instance called testdb2 that was previously cataloged in the node directory:

```
db2 attach to testdb2
```

After performing maintenance activities for the testdb2 instance, detach from an instance:

```
db2 detach
```

Working with instances on the same or different DB2 copies

You can run multiple instances concurrently, in the same DB2 copy or in different DB2 copies.

About this task

To prevent one instance from accessing the database of another instance, the database files for an instance are created under a directory that has the same name as the instance name. For example, when creating a database on drive `C:` for instance `DB2`, the database files are created inside a directory called `C:\DB2`. Similarly, when creating a database on drive `C:` for instance `TEST`, the database files are created inside a directory called `C:\TEST`. By default, its value is the drive letter where DB2 product is installed. For more information, see the **dftdbpath** database manager configuration parameter.

Procedure

- To work with instances in the same DB2 copy, you must:
  1. Create or upgrade all instances to the same DB2 copy.
  2. Set the **DB2INSTANCE** environment variable to the name of the instance you are working with. This action must occur before you issue commands against the instance.
- To work with an instance in a system with multiple DB2 copies, use either of the following methods:
  - Use the Command window from the **Start** > **Programs** > **IBM DB2** > ***DB2 Copy Name*** > **Command Line Tools** > **Command Window**. The Command window is already set up with the correct environment variables for the particular DB2 copy chosen.
  - Use `db2envar.bat` from a Command window:
    1. Open a Command window.
    2. Run the `db2envar.bat` file using the fully qualified path for the DB2 copy that you want the application to use:

       ```
       DB2_Copy_install_dir\bin\db2envar.bat
       ```

Stopping instances (Linux, UNIX)

You might need to stop the current instance of the database manager.

Before you begin

1. Log in or attach to an instance with a user ID or name that has SYSADM, SYSCTRL, or SYSMAINT authority on the instance; or, log in as the instance owner.
2. Display all applications and users that are connected to the specific database that you want to stop. To ensure that no vital or critical applications are running, use the LIST APPLICATIONS command.
3. Force all applications and users off the database by using the FORCE APPLICATION command.
4. If command line processor sessions are attached to an instance, you must run the **TERMINATE** command to end each session before running the **db2stop** command.

About this task

When you run commands to start or stop an instance, the DB2 database manager applies the command to the current instance. For more information, see "Identifying the current instance" on page 30.

Restrictions

The **db2stop** command can be run only at the server.

No database connections are allowed when running this command; however, if there are any instance attachments, they are forced off before the instance is stopped.

Procedure

To stop an instance on a Linux or UNIX operating system:

- From the command line, enter the **db2stop** command. The DB2 database manager applies the command to the current instance.
- From IBM Data Studio, open the task assistant for stopping the instance. For more information, see IBM Data Studio: Administering databases with task assistants.

Stopping instances (Windows)

You might need to stop the current instance of the database manager.

Before you begin

1. The user account stopping the DB2 database service must have the correct privilege as defined by the Windows operating system. The user account can be a member of the Administrators, Server Operators, or Power Users group.
2. Display all applications and users that are connected to the specific database that you want to stop. To ensure that no vital or critical applications are running, use the LIST APPLICATIONS command.
3. Force all applications and users off the database by using the FORCE APPLICATION command.
4. If command line processor sessions are attached to an instance, you must run the **TERMINATE** command to end each session before running the **db2stop** command.

About this task

Note: When you run commands to start or stop an instance, the database manager applies the command to the current instance. For more information, see "Identifying the current instance" on page 30.

Restrictions

The **db2stop** command can be run only at the server.

No database connections are allowed when running this command; however, if there are any instance attachments, they are forced off before the DB2 database service is stopped.

When you are using the database manager in a partitioned database environment, each database partition server is started as a service. To stop an instance, all services must be stopped.

Procedure

To stop the instance:

- From the command line, enter the **db2stop** command. The DB2 database manager applies the command to the current instance.
- From the command line, enter the **NET STOP** command.
- From IBM Data Studio, open the task assistant for stopping the instance. For more information, see IBM Data Studio: Administering databases with task assistants.

Upgrading instances

The following upgrade instance tasks are one step in the task for upgrading DB2 servers. For more information, see “Upgrading DB2 servers” in the upgrade documentation at http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.qb.upgrade.doc/doc/c0011933.html.

Upgrading DB2 Version 9.5 or DB2 Version 9.7 instances

As part of the overall process of upgrading your DB2 database server to DB2 Version 10.1, you must upgrade your instances.

Before you begin

- You must have root user authority on Linux and UNIX operating systems or Local Administrator authority on Windows.
- You must install any DB2 database add-on products that were installed in the DB2 copy from which you are upgrading.
- Before running the **db2iupgrade** command, the following steps are recommended:
  - Verify that databases are ready for DB2 upgrade. This step is important in partitioned database environments because the **db2ckupgrade** command might return an error in one database partition and cause the instance upgrade to fail.
  - On Linux and UNIX operating systems, ensure that there is 5GB of free space in the `/tmp` directory. The instance upgrade trace file is written to `/tmp`.
  - Gather pre-upgrade diagnostic information to help diagnose any problem that might occur after the upgrade.

About this task

On Linux and UNIX operating systems, you must manually upgrade your instances. On Windows operating systems, you must manually upgrade them if you did not choose to automatically upgrade your existing DB2 copy during the DB2 Version 10.1 installation.

Restriction

- On Linux and UNIX operating systems, you must not set up the instance environment for the root user. Running the **db2iupgrade** or the **db2icrt** command when you set up the instance environment is not supported.
- For additional restrictions on instance upgrade, review “Upgrade restrictions for DB2 servers” in *Upgrading to DB2 Version 10.1*.
- You must be upgrading from DB2 Version 9.5 or DB2 Version 9.7.

Procedure

To manually upgrade your existing instances to DB2 Version 10.1 using the **db2iupgrade** command:

1. Determine if you can upgrade your existing instances to a DB2 Version 10.1 copy that you installed by performing the following actions:
   - Determine the node type. The following examples show how to use the **GET DBM CFG** command to find out the node type:

Operating system	Examples

Linux and UNIX	`db2 GET DBM CFG | grep 'Node type'`
	`Node type = Partitioned database server with local and remote clients`

Windows	`db2 GET DBM CFG | find  "Node type"`
	`Node type = Partitioned database server with local and remote clients`

If you cannot upgrade your instance to any DB2 Version 10.1 copy that you installed, you must install a copy of the DB2 Version 10.1 database product that supports upgrade of your instance type before you can proceed with the next step.

2. Disconnect all users, stop back end processes, and stop your existing instances by running the following command:

```
db2stop force (Disconnects all users and stops the instance)
db2 terminate (Terminates back-end process)
```

3. Log on to the DB2 database server with root user authority on Linux and UNIX operating systems or Local Administrator authority on Windows operating systems.
4. Upgrade your existing instances by running the **db2iupgrade** command from the target DB2 Version 10.1 copy location. The **db2iupgrade** command only needs to be run on the instance owning node. The following table shows how to run the **db2iupgrade** command to upgrade your instances:

Operating system	Command syntax
Linux and UNIX	*$DB2DIR*/instance/db2iupgrade [-u *fencedID*] *InstName*[a]
Windows	"%**DB2PATH**%"\bin\db2iupgrade *InstName* /u:*user*,*password*[b]

Note:

a. Where *DB2DIR* is set to the location you specified during DB2 Version 10.1 installation, *fencedID* is the user name under which the fenced user-defined functions (UDFs) and stored procedures will run, and *InstName* is the login name of the instance owner. This example upgrades the instance to the highest level for DB2 database product that you installed, use the **-k** option if you want to keep the pre-upgrade instance type.

b. Where **DB2PATH** is set to the location you specified during DB2 Version 10.1 installation, *user* and *password* are the user name and password under which the DB2 service will run, and *InstName* is the name of the instance.

If you did not install all DB2 database add-on products that were installed in the DB2 copy from which you are upgrading, the instance upgrade fails and returns a warning message. If you plan to install these products later on or you no longer need the functionality provided by these products, use the **-F** parameter to upgrade the instance.

The **db2iupgrade** command calls the **db2ckupgrade** command with the **-not1** parameter to verify that the local databases are ready for grade. The `update.log` is specified as the log file for **db2ckupgrade**, and the default log file created for **db2iupgrade** is `/tmp/db2ckupgrade.log.processID`. On Linux and UNIX operating systems, the log file is created in the instance home directory. On Windows operating systems, the log file is created in the current directory where you are running the **db2iupgrade** command. The **-not1** parameter disables the check for type-1 indexes. Verify that you do not have type-1 indexes in your databases before upgrading the instance. The **db2iupgrade** does not run as long as the **db2ckupgrade** command reports errors. Check the log file if you encounter any errors.

5. Log on to the DB2 database server as a user with sufficient authority to start your instance.
6. Restart your instance by running the **db2start** command:

   ```
   db2start
   ```
7. Verify that your instance is running on to DB2 Version 10.1 by running the **db2level** command:

   ```
   db2level
   ```

   The Informational tokens should include a string like "DB2 Version 10.1.*X*.*X*" where *X* is a digit number.

Upgrading DB2 Version 9.8 instances

As part of the overall process of upgrading your DB2 database server to DB2 Version 10.1, you must upgrade your Version 9.8 instances.

Before you begin

- Your DB2 Version 9.8 instance must be a DB2 pureScale instance.
- You must have root user authority on Linux and UNIX operating systems.
- You must install any DB2 database add-on products that were installed in the DB2 copy from which you are upgrading.
- Before running the **db2iupgrade** command, the following steps are recommended:
  - Verify that databases are ready for DB2 upgrade. This step is important in DB2 pureScale environments because the **db2ckupgrade** command might return an error in one member and cause the instance upgrade to fail.
  - On Linux and UNIX operating systems, ensure that there is 5GB of free space in the `/tmp` directory. The instance upgrade trace file is written to `/tmp`.
  - Gather pre-upgrade diagnostic information to help diagnose any problem that might occur after the upgrade.

About this task

On Linux and UNIX operating systems, you must manually upgrade your DB2 pureScale instances from Version 9.8.

Restrictions

- On Linux and UNIX operating systems, you must not set up the instance environment for the root user. Running the **db2iupgrade** or the **db2icrt** command when you set up the instance environment is not supported.
- For additional restrictions on instance upgrade, review “Upgrade restrictions for DB2 servers” in *Upgrading to DB2 Version 10.1*.

Procedure

To manually upgrade your existing Version 9.8 instances to DB2 Version 10.1 using the **db2iupgrade** command:

1. Log on to the DB2 server with root user authority.
2. Upgrade your existing Version 9.8 instances by issuing the **db2iupgrade** command from the target DB2 Version 10.1 copy location. You should issue the

db2iupgrade command from the Version 10.1 installation path from all the members first and then from the CFs. The following example shows how to use this command:

```
$DB2DIR/instance/db2iupgrade [ -u fencedID ] InstName
```

Where *DB2DIR* is set to the location that you specified during DB2 Version 10.1 installation, *fencedID* is the user name under which the fenced user-defined functions (UDFs) and stored procedures will run, and *InstName* is the login name of the instance owner.

If you did not install all DB2 database add-on products that were installed in the DB2 copy from which you are upgrading, the instance upgrade fails and returns a warning message. If you plan to install these products later on or you no longer need the functionality provided by these products, use the **-F** parameter to upgrade the instance.

Note: You must stop the Version 9.8 instance using the **db2stop** command before issuing the **db2iupgrade** command. If you do not stop Version 9.8 instance before using the **db2iupgrade** command, your instance upgrade might fail.

3. Log on to the DB2 database server as a user with sufficient authority to start your instance.
4. Restart the DB2 instance on all members and CFs with updated resources for the cluster management software and the cluster file system software by issuing the **db2start instance on <hostname>** command, and then issue the **db2start** command. If you find inconsistencies between the cluster manager resource model and the db2nodes.cfg repair the cluster manager resources by using the **db2cluster -cm -repair -resources** command.
5. Verify that your instances are running on to DB2 Version 10.1 by running the **db2level** command: The Informational tokens should include a string like "DB2 Version 10.1.*X*.*X*" where *X* is a digit number.
6. Rebuild the contents of the network resiliency condition and response resources in the cluster by issuing the **db2cluster -cfs -repair -network_resiliency -all** command.

What to do next

After upgrading your Version 9.8 DB2 pureScale instance, you must upgrade your database. For more details. see “Upgrading databases” in *Upgrading to DB2 Version 10.1*.

Dropping instances

To drop a root instance, issue the **db2idrop** command. To drop non-root instances, you must uninstall your DB2 database product.

Procedure

To remove a root instance using the command line:

1. Stop all applications that are currently using the instance.
2. Stop the Command Line Processor by running **terminate** commands in each Command window.
3. Stop the instance by running the **db2stop** command.
4. Back up the instance directory indicated by the **DB2INSTPROF** registry variable.

On Linux and UNIX operating systems, consider backing up the files in the *INSTHOME*/sqllib directory (where *INSTHOME* is the home directory of the instance owner). For example, you might want to save the database manager configuration file, db2systm, the db2nodes.cfg file, user-defined functions (UDFs), or fenced stored procedure applications.

5. For Linux and UNIX operating systems only, log off as the instance owner and log in as a user with root user authority.
6. Issue the **db2idrop** command. For example:

   ```
   db2idrop InstName
   ```

   where *InstName* is the name of the instance being dropped.

   The **db2idrop** command removes the instance entry from the list of instances and removes the sqllib subdirectory under the instance owner's home directory.

   Note: On Linux and UNIX operating systems, if you issue the **db2idrop** command and receive a message stating that the *INSTHOME*/sqllib subdirectory cannot be removed, one reason could be that the *INSTHOME*/adm subdirectory contains files with the .nfs extension. The adm subdirectory is an NFS-mounted system and the files are controlled on the server. You must delete the *.nfs files from the file server from where the directory is being mounted. Then you can remove the *INSTHOME*/sqllib subdirectory.
7. For Windows operating systems, if the instance that you dropped was the default instance, set a new default instance by issuing the **db2set** command:

   ```
   db2set db2instdef=instance_name -g
   ```

   where *instance_name* is the name of an existing instance.
8. For Linux and UNIX operating systems, remove the instance owner's user ID and group (if used only for that instance). Do not remove these if you are planning to re-create the instance.

   This step is optional since the instance owner and the instance owner group might be used for other purposes.

Chapter 2. Configuring intances

To achieve maximum performance, use the database manager configuration parameters to allocate enough disk space and memory for your instance. The default values of the parameters might be sufficient to meet your needs but might not exploit all the resources available in your environment.

Configuration parameters

When you create a DB2 database instance or a database, a configuration file is created with default parameter values. You can modify these parameter values to improve performance and other characteristics of the instance or database.

The disk space and memory allocated by the database manager on the basis of default values of the parameters might be sufficient to meet your needs. In some situations, however, you might not be able to achieve maximum performance using these default values.

Configuration files contain parameters that define values such as the resources allocated to the DB2 database products and to individual databases, and the diagnostic level. There are two types of configuration files:

- The database manager configuration file for each DB2 instance
- The database configuration file for each individual database.

The *database manager configuration file* is created when a DB2 instance is created. The parameters it contains affect system resources at the instance level, independent of any one database that is part of that instance. Values for many of these parameters can be changed from the system default values to improve performance or increase capacity, depending on your system's configuration.

There is one database manager configuration file for each client installation as well. This file contains information about the client enabler for a specific workstation. A subset of the parameters available for a server are applicable to the client.

Database manager configuration parameters are stored in a file named `db2systm`. This file is created when the instance of the database manager is created. In Linux and UNIX environments, this file can be found in the `sqllib` subdirectory for the instance of the database manager. In Windows, the default location of this file varies from edition to edition of the Windows family of operating systems. You can verify the default directory on Windows, check the setting of the **DB2INSTPROF** registry variable using the command **db2set DB2INSTPROF**. You can also change the default instance directory by changing the **DB2INSTPROF** registry variable. If the **DB2INSTPROF** variable is set, the file is in the `instance` subdirectory of the directory specified by the **DB2INSTPROF** variable.

Other profile-registry variables that specify where run-time data files should go should query the value of **DB2INSTPROF**. This includes the following variables:

- **DB2CLIINIPATH**
- **diagpath**
- **spm_log_path**

All database configuration parameters are stored in a file named SQLDBCONF. You cannot directly edit these files. You can only change or view these files via a supplied API or by a tool which calls that API.

In a partitioned database environment, this file resides on a shared file system so that all database partition servers have access to the same file. The configuration of the database manager is the same on all database partition servers.

Most of the parameters either affect the amount of system resources that are allocated to a single instance of the database manager, or they configure the setup of the database manager and the different communications subsystems based on environmental considerations. In addition, there are other parameters that serve informative purposes only and cannot be changed. All of these parameters have global applicability independent of any single database stored under that instance of the database manager.

A *database configuration file* is created when a database is created, and resides where that database resides. There is one configuration file per database. Its parameters specify, among other things, the amount of resource to be allocated to that database. Values for many of the parameters can be changed to improve performance or increase capacity. Different changes might be required, depending on the type of activity in a specific database.

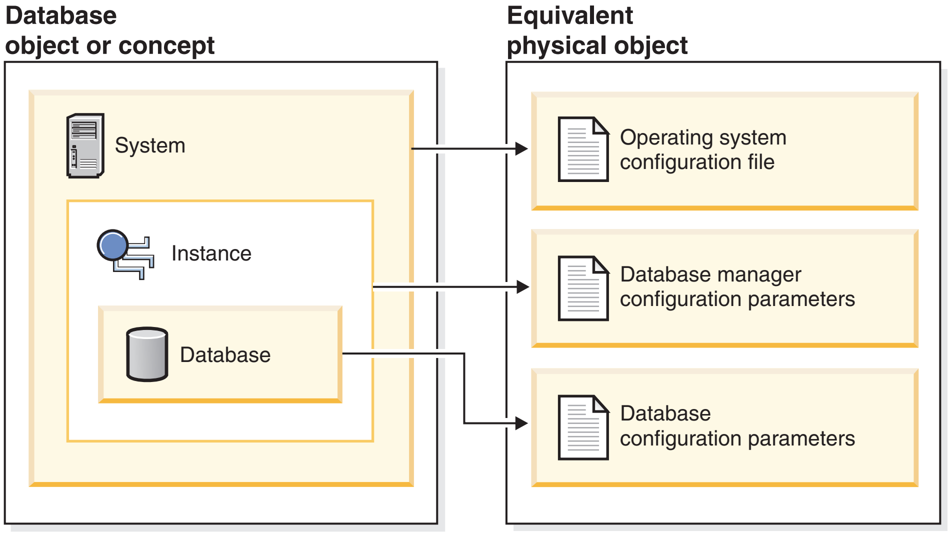

Database object or concept
Equivalent physical object
System
Instance
Database
Operating system configuration file
Database manager configuration parameters
Database configuration parameters

Figure 2. Relationship between database objects and configuration files

Configuring instances with database manager configuration parameters

The disk space and memory allocated by the database manager based on default values of the parameters might be sufficient to meet your needs. In some situations, however, you might not be able to achieve maximum performance by using these default values.

About this task

Since the default values are oriented towards machines that have relatively small memory resources and are dedicated as database servers, you might need to modify these values if your environment has:

- Large databases
- Large numbers of connections
- High performance requirements for a specific application
- Unique query or transaction loads or types

Each transaction processing environment is unique in one or more aspects. These differences can have a profound impact on the performance of the database manager when using the default configuration. For this reason, you are strongly advised to tune your configuration for your environment.

A good starting point for tuning your configuration is to use the Configuration Advisor or the **AUTOCONFIGURE** command. These tools generate values for parameters based on your responses to questions about workload characteristics.

Some configuration parameters can be set to `AUTOMATIC`, allowing the database manager to automatically manage these parameters to reflect the current resource requirements. To turn off the `AUTOMATIC` setting of a configuration parameter while maintaining the current internal setting, use the **MANUAL** keyword with the **UPDATE DATABASE CONFIGURATION** command. If the database manager updates the value of these parameters, the **GET DB CFG SHOW DETAIL** and **GET DBM CFG SHOW DETAIL** commands will show the new value.

Parameters for an individual database are stored in a configuration file named `SQLDBCONF`. This file is stored along with other control files for the database in the `SQL`*nnnnn* directory, where *nnnnn* is a number assigned when the database was created. Each database has its own configuration file, and most of the parameters in the file specify the amount of resources allocated to that database. The file also contains descriptive information, as well as flags that indicate the status of the database.

Attention: If you edit `db2systm`, `SQLDBCON`, or `SQLDBCONF` by using a method other than those provided by the database manager, you might make the database unusable. Do not change these files by using methods other than those documented and supported by the database manager.

In a partitioned database environment, a separate `SQLDBCONF` file exists for each database partition. The values in the `SQLDBCONF` file might be the same or different at each database partition, but the recommendation is that in a homogeneous environment, the configuration parameter values should be the same on all database partitions. Typically, there could be a catalog node needing different database configuration parameters setting, while the other data partitions have different values again, depending on their machine types, and other information.

Procedure

1. Update configuration parameters.
   - Using the command line processor:

     Commands to change the settings can be entered as follows:

     For database manager configuration parameters:
     - **GET DATABASE MANAGER CONFIGURATION** (or **GET DBM CFG**)
     - **UPDATE DATABASE MANAGER CONFIGURATION** (or **UPDATE DBM CFG**)
     - **RESET DATABASE MANAGER CONFIGURATION** (or **RESET DBM CFG**) to reset *all* database manager parameters to their default values
     - **AUTOCONFIGURE**

For database configuration parameters:

- **GET DATABASE CONFIGURATION** (or **GET DB CFG**)
- **UPDATE DATABASE CONFIGURATION** (or **UPDATE DB CFG**)
- **RESET DATABASE CONFIGURATION** (or **RESET DB CFG**) to reset *all* database parameters to their default values
- **AUTOCONFIGURE**

- Using application programming interfaces (APIs):

  The APIs can be called from an application or a host-language program. Call the following DB2 APIs to view or update configuration parameters:
  - db2AutoConfig - Access the Configuration Advisor
  - db2CfgGet - Get the database manager or database configuration parameters
  - db2CfgSet - Set the database manager or database configuration parameters
- Using common SQL application programming interface (API) procedures:

  You can call the common SQL API procedures from an SQL-based application, a DB2 command line, or a command script. Call the following procedures to view or update configuration parameters:
  - GET_CONFIG - Get the database manager or database configuration parameters
  - SET_CONFIG - Set the database manager or database configuration parameters
- Using IBM Data Studio, right-click the instance to open the task assistant to update the database manager configuration parameters.

2. View updated configuration values.

   For some database manager configuration parameters, the database manager must be stopped (**db2stop**) and then restarted (**db2start**) for the new parameter values to take effect.

   For some database parameters, changes take effect only when the database is reactivated, or switched from offline to online. In these cases, all applications must first disconnect from the database. (If the database was activated, or switched from offline to online, then it must be deactivated and reactivated.) Then, at the first new connect to the database, the changes take effect.

   If you change the setting of a configurable online database manager configuration parameter while you are attached to an instance, the default behavior of the **UPDATE DBM CFG** command is to apply the change immediately. If you do not want the change applied immediately, use the **DEFERRED** option on the **UPDATE DBM CFG** command.

   To change a database manager configuration parameter online:

   ```
   db2 attach to instance-name
   db2 update dbm cfg using parameter-name value
   db2 detach
   ```

   For clients, changes to the database manager configuration parameters take effect the next time the client connects to a server.

   If you change a configurable online database configuration parameter while connected, the default behavior is to apply the change online, wherever possible. Note that some parameter changes might take a noticeable amount of time to take effect due to the additional processing time associated with allocating space. To change configuration parameters online from the command line processor, a connection to the database is required. To change a database configuration parameter online:

```
db2 connect to dbname
db2 update db cfg using parameter-name parameter-value
db2 connect reset
```

Each configurable online configuration parameter has a *propagation class* associated with it. The propagation class indicates when you can expect a change to the configuration parameter to take effect. There are four propagation classes:

- **Immediate**: Parameters that change immediately upon command or API invocation. For example, **diaglevel** has a propagation class of immediate.
- **Statement boundary**: Parameters that change on statement and statement-like boundaries. For example, if you change the value of **sortheap**, all new requests use the new value.
- **Transaction boundary**: Parameters that change on transaction boundaries. For example, a new value for **dl_expint** is updated after a COMMIT statement.
- **Connection**: Parameters that change on new connection to the database. For example, a new value for **dft_degree** takes effect for new applications connecting to the database.

While new parameter values might not be immediately effective, viewing the parameter settings (by using the **GET DATABASE MANAGER CONFIGURATION** or **GET DATABASE CONFIGURATION** command) always shows the latest updates. Viewing the parameter settings by using the **SHOW DETAIL** clause on these commands shows both the latest updates and the values in memory.

3. Rebind applications after updating database configuration parameters.

   Changing some database configuration parameters can influence the access plan chosen by the SQL and XQuery optimizer. After changing any of these parameters, consider rebinding your applications to ensure that the best access plan is being used for your SQL and XQuery statements. Any parameters that were modified online (for example, by using the **UPDATE DATABASE CONFIGURATION IMMEDIATE** command) cause the SQL and XQuery optimizer to choose new access plans for new query statements. However, the query statement cache is not purged of existing entries. To clear the contents of the query cache, use the FLUSH PACKAGE CACHE statement.

   Note: A number of configuration parameters (for example, **health_mon**) are described as having acceptable values of either Yes or No, or On or Off in the help and other DB2 documentation. To clarify, Yes should be considered equivalent to On and No should be considered equivalent to Off.

Environment variables and the profile registries

Environment and registry variables control your DB2 database environment. Use the DB2 profile registries to view and update information about variables and instances.

Before the DB2 database profile registries were introduced, setting environment variables required you to specify a value for an environment variable and restart your computer. You can now use the DB2 profile registries to control most variables that affect your DB2 database environment.

Use the profile registries to control the environment variables from one computer. Different levels of support are provided through the different profiles. You can administer the environment variables remotely by using the DB2 administration server.

A DB2 database is affected by the following profile registries:

- The DB2 instance-level profile registry contains registry variables for an instance. Values that are defined in this registry override their settings in the global registry.
- The DB2 global-level profile registry contains settings that are used if a registry variable is not set for an instance. All instances that pertain to a particular copy of DB2 Enterprise Server Edition can access this registry.
- The DB2 instance node-level profile registry contains variable settings that are specific to a database partition in a partitioned database environment. Values that are defined in this registry override their settings at the instance and global levels.
- The DB2 user-level profile registry contains settings that are specific to each user. Values that are defined in this registry override their settings in the other registries.

The DB2 database system configures the operating environment by checking for registry values and environment variables and resolving them in the following order:

1. Environment variables that are set outside the profile registries.
2. Registry variables that are set with the user-level profile.
3. Registry variables that are set with the instance node-level profile.
4. Registry variables that are set with the instance-level profile.
5. Registry variables that are set with the global-level profile.

The DB2 instance profile registry contains a list of all instances that are associated with the current copy. A list exists for each DB2 copy. You can see the complete list of all the instances that are available on the system by running the **db2ilist** command. This profile registry does not contain variable settings.

Profile registry locations and authorization requirements

The DB2 profile registries have different locations and authorization requirements on each operating system. Authorization is required to update the values of variables in each profile registry.

Table 1. Profile registry locations and authorization requirements

Profile registry	Location on Windows	Location on Linux and UNIX	Linux and UNIX authorization requirements	Windows authorization requirements
Instance-level profile registry	`\HKEY_LOCAL_computer \SOFTWARE\IBM\DB2 \PROFILES\` *instance_name*	*instance_home*/sqllib/profile.env where *instance_home* is the home path of the instance owner.	-rw-rw-r-- *instance_owner* *instance_owner_group* profile.env	You must be a member of the DB2 administrators group (DB2ADMNS).
Global-level profile registry	`\HKEY_LOCAL_computer \SOFTWARE\IBM\DB2 \GLOBAL_PROFILE`	For root installations:/var/db2/global.reg For non-root installations: *home_directory*/sqllib/global.reg	To modify a global registry variable in root installations, you must be logged on with root authority.	You must be a member of the DB2 administrators group (DB2ADMNS).

Table 1. Profile registry locations and authorization requirements (continued)

Profile registry	Location on Windows	Location on Linux and UNIX	Linux and UNIX authorization requirements	Windows authorization requirements

Instance node-level profile registry	`...\SOFTWARE\IBM\DB2\PROFILES\`*instance_name*`\NODES\`*node_number*	*instance_home*`/sqllib/nodes/`*node_number*`.env`
		where *instance_home* is the home path of the instance owner.
			For the directory that contains the file:
			`drwxrwsr-w` *instance_owner* *instance_owner_group* `nodes`
			For the file:
			`-rw-rw-r--` *instance_owner* *instance_owner_group* *node_number*`.env`
				You must be a member of the DB2 administrators group (DB2ADMNS).

User-level profile registry	The Lightweight Directory Access Protocol (LDAP) directory.	Does not apply.	Does not apply.	You must be a member of the DB2 administrators group (DB2ADMNS).
Instance profile registry	`\HKEY_LOCAL_computer\SOFTWARE\IBM\DB2\PROFILES\`*instance_name*	For root installations:`/var/db2/global.reg` For non-root installations: *home_directory*`/sqllib/global.reg`	None required.	None required.

Setting registry and environment variables

Most environment variables are set in the DB2 database profile registries by using the **db2set** command. The few variables that are set outside the profile registries require different commands depending on your operating system.

Before you begin

Ensure that you have the privileges that are required to set registry variables.

On Linux and UNIX operating systems, you must have the following privileges:

- SYSADM authority to set variables in the instance-level registry
- root authority to set variables in the global-level registry

On Windows operating systems, you must have one of the following privileges:

- local Administrator authority
- SYSADM authority with the following conditions:
  - If extended security is enabled, SYSADM users must belong to the DB2ADMNS group.
  - If extended security is not enabled, SYSADM users can make updates if the appropriate permissions are granted to them in the Windows registry.

About this task

When you use the **db2set** command to set variables in the profile registries, you do not need to restart your computer for variable values to take effect. However, changes do not affect DB2 applications that are currently running or users that are active. The DB2 registry applies the updated information to DB2 server instances and DB2 applications that are started after the changes are made.

If DB2 variables are set outside the registry, you cannot administer those variables remotely. Also, you must restart the computer for the variable values to take effect.

The **DB2INSTANCE** and **DB2NODE** DB2 environment variables are not stored in the DB2 profile registries. See the topics about setting environment variables outside the profile registries for information about setting these variables.

On Linux and UNIX operating systems, the instance-level profile registry is stored in the `profile.env` text file. If two or more users set a registry variable with the **db2set** command at almost the same time, the size of this file is reduced to zero. Also, the output from the **db2set -all** command displays inconsistent values.

Procedure

To set a registry variable:

Issue the **db2set** command with the relevant parameters.
The following table shows some of the ways that you can set registry variables with the **db2set** command. See the **db2set** command reference topic for more information about the parameters and usage of this command.

Table 2. Common commands for setting registry variables

Desired Action	Command
Set a registry variable for the current or default instance.	`db2set` *registry_variable_name=new_value*
Set a registry variable for all databases in an instance.	`db2set` *registry_variable_name=new_value* `-i` *instance_name*
Set a registry variable for a particular database partition in an instance.	`db2set` *registry_variable_name=new_value* `-i` *instance_name* *database_partition_number*
Set a registry variable for all instances that pertain to a DB2 Enterprise Server Edition installation.	`db2set` *registry_variable_name=new_value* `-g`
Set a registry variable at the user level in a Lightweight Directory Access Protocol (LDAP) environment.	`db2set` *registry_variable_name=new_value* `-ul`
Set a registry variable at the global level in an LDAP environment. **DB2LDAP_KEEP_CONNECTION** and **DB2LDAP_SEARCH_SCOPE** are the only two registry variables that can be set at the LDAP global level.	`db2set` *registry_variable_name=new_value* `-gl`

Tip: If a registry variable requires Boolean values as arguments, the values `YES`, `1`, `TRUE`, and `ON` are all equivalent and the values `NO`, `0`, `FALSE`, and `OFF` are also equivalent. For any variable, you can specify any of the appropriate equivalent values.

Setting environment variables outside the profile registries on Linux and UNIX operating systems

On Linux and UNIX operating systems, you must set the **DB2INSTANCE** system environment variable outside the profile registries. If you want to set any other variables, those variables must be set in one or more of the profile registries.

About this task

You can use the scripts db2profile (for Bourne or Korn shell) and db2cshrc (for C shell) to set the **DB2INSTANCE** variable to an instance name that you specify. The scripts are in the *instance_home*/sqllib directory, where *instance_home* is the home directory of the instance owner.

Instance owners and users with SYSADM privileges can customize these scripts for all users of an instance. Alternatively, users can copy and customize a script, then invoke a script directly or add it to their .profile or .login files.

To set variables that are not supported by the DB2 database manager, define them in the userprofile and usercshrc script files. These files are also in the *instance_home*/sqllib directory.

Procedure

To set an environment variable outside the profile registries:

Set an environment variable by using one of the following methods:

Option	Description
Set the environment variable at the instance level for a Bourne or Korn shell.	Run the db2profile script.
Set the environment variable at the instance level for a C shell.	Run the db2cshrc script.
Set the environment variable for the current session for a Bourne shell.	Issue the following command: export *env_variable_name=new_value*
Set the environment variable for the current session for a C shell.	Issue the following command: setenv *env_variable_name new_value*
Set the environment variable for the current session for a Korn shell.	Issue the following command: *environment_variable_name=new_value* export *environment_variable_name*

Setting environment variables outside the profile registries on Windows

On Windows operating systems, the **DB2INSTANCE**, **DB2NODE**, and **DB2PATH** environment variables can be set only outside the profile registries. You are required to set only the **DB2PATH** variable.

About this task

On Windows operating systems, the following environment variables are set outside the profile registries:

- The **DB2INSTANCE** environment variable specifies the instance that is active by default. If this variable is not set, the DB2 database manager uses the value of the **DB2INSTDEF** variable as the current instance.
- The **DB2NODE** environment variable specifies the target logical node of a database partition server to which requests are routed.
- The **DB2PATH** environment variable specifies the directory where the DB2 database product is installed on Windows 32-bit operating systems.

If you want to set any other variables, those variables must be set in one or more of the profile registries.

You can determine the value of an environment variable by using the **echo** command. For example, to check the value of the **DB2NODE** environment variable, issue the following command:

```
echo %db2path%
```

Procedure

To set an environment variable outside the profile registries:

Set an environment variable by using one of the following options.

Option	Description
Set the environment variable at the instance level.	1. Follow the appropriate procedure for your Windows operating system. 2. Restart your computer.

Set the environment variable for the current session.	Issue the following command:
	`set env_variable_name=new_value`
	`db2start`

Set the environment variable for the current session for a C shell.	Issue the following command: `setenv env_variable_name new_value`

Identifying the current instance

Most commands that you issue or configuration changes that you make apply by default to the current instance. You can identify the current instance by checking the values of certain environment variables.

About this task

When you run commands to start or stop the database manager for an instance, the database manager applies the command to the current instance. To determine the current instance, the database manager checks the values of certain environment variables in the following order:

1. The value of the **DB2INSTANCE** environment variable for the current session.
2. The value of the **DB2INSTANCE** system environment variable.
3. On Windows operating systems, the value of the **DB2INSTDEF** registry variable.

Procedure

To identify the current instance:

Check the value of the relevant environment variable.

Option	Description
View the value of the `DB2INSTANCE` environment variable for the current session.	Issue the following command: `db2 get instance`

View the value of the `DB2INSTANCE` system environment variable.	• On Windows operating systems, issue the following command:
	`echo %DB2INSTANCE%`
	• On Linux and UNIX operating systems, issue the following command:
	`echo $DB2INSTANCE`

View the value of the `DB2INSTDEF` registry variable.	Issue the following command: `db2set DB2INSTDEF`

Setting variables at the instance level in a partitioned database environment

In a partitioned database environment, the way that you set registry variables in the instance-level profile registry depends on your operating system.

About this task

On Linux and UNIX operating systems, the instance-level profile registry is stored in a text file in the `sqllib` directory. Because the `sqllib` directory is on a file system that is shared by all physical database partitions, you can set a registry variable from any database partition.

On Windows operating systems, the DB2 database manager stores the instance-level profile registry in the Windows registry. As a result, data is not shared across physical database partitions. To set a variable for all database partitions, you must use the **rah** command to ensure that the command that you use to set the variable is run on all computers. If you set a registry variable from a database partition and do not use the **rah** command, the variable is set only for that database partition in the current instance.

You can also use the **DB2REMOTEPREG** registry variable to configure a computer that is not the instance owner to use the values of registry variables on the instance-owning computer.

Procedure

To set a registry variable for all database partitions of the current instance:

Issue the command for your operating system from any database partition.

- On Linux and UNIX operating systems, issue the following command:
  ```
  db2set registry_variable_name=new_value
  ```
- On Windows operating systems, issue the following command:
  ```
  rah db2set registry_variable_name=new_value
  ```

Chapter 3. Autonomic computing

The DB2 autonomic computing environment is self-configuring, self-healing, self-optimizing, and self-protecting. By sensing and responding to situations that occur, autonomic computing shifts the burden of managing a computing environment from database administrators to technology. These capabilities for DB2 autonomic computing are provided by the automatic features that are described in the following section.

Automatic features

Automatic features assist you in managing your database system. They allow your system to perform self-diagnosis and to anticipate problems before they happen by analyzing real-time data against historical problem data. You can configure some of the automatic tools to make changes to your system without intervention to avoid service disruptions.

When you create a database, some of the following automatic features are enabled by default, but others you must enable manually:

Self-tuning memory (single-partition databases only)
The self-tuning memory feature simplifies the task of memory configuration. This feature responds to significant changes in workload by automatically and iteratively adjusting the values of several memory configuration parameters and the sizes of the buffer pools, thus optimizing performance. The memory tuner dynamically distributes available memory resources among several memory consumers, including the sort function, the package cache, the lock list, and buffer pools. You can disable self-tuning memory after creating a database by setting the database configuration parameter **self_tuning_mem** to `OFF`.

Automatic storage
The automatic storage feature simplifies storage management for table spaces. When you create a database, you specify the storage paths for the default storage group where the database manager places your table space data. Then, the database manager manages the container and space allocation for the table spaces as you create and populate them. You can then also create new storage groups or alter existing ones.

Data compression
Both tables and indexes can be compressed to save storage. Compression is fully automatic; once you specify that a table or index should be compressed using the COMPRESS YES clause of the CREATE TABLE, ALTER TABLE, CREATE INDEX or ALTER INDEX statements, there is nothing more you must do to manage compression. (Converting an existing uncompressed table or index to be compressed does require a REORG to compress existing data). Temporary tables are compressed automatically; indexes for compressed tables are also compressed automatically, by default.

Automatic database backups
A database can become unusable due to a wide variety of hardware or software failures. Ensuring that you have a recent, full backup of your database is an integral part of planning and implementing a disaster recovery strategy for your system. Use automatic database backups as part

of your disaster recovery strategy to enable the database manager to back up your database both properly and regularly.

Automatic reorganization
After many changes to table data, the table and its indexes can become fragmented. Logically sequential data might reside on nonsequential pages, forcing the database manager to perform additional read operations to access data. The automatic reorganization process periodically evaluates tables and indexes that have had their statistics updated to see if reorganization is required, and schedules such operations whenever they are necessary.

Automatic statistics collection
Automatic statistics collection helps improve database performance by ensuring that you have up-to-date table statistics. The database manager determines which statistics are required by your workload and which statistics must be updated. Statistics can be collected either asynchronously (in the background) or synchronously, by gathering runtime statistics when SQL statements are compiled. The DB2 optimizer can then choose an access plan based on accurate statistics. You can disable automatic statistics collection after creating a database by setting the database configuration parameter **auto_runstats** to OFF. Real-time statistics gathering can be enabled only when automatic statistics collection is enabled. Real-time statistics gathering is controlled by the **auto_stmt_stats** configuration parameter.

Configuration Advisor
When you create a database, this tool is automatically run to determine and set the database configuration parameters and the size of the default buffer pool (IBMDEFAULTBP). The values are selected based on system resources and the intended use of the system. This initial automatic tuning means that your database performs better than an equivalent database that you could create with the default values. It also means that you will spend less time tuning your system after creating the database. You can run the Configuration Advisor at any time (even after your databases are populated) to have the tool recommend and optionally apply a set of configuration parameters to optimize performance based on the current system characteristics.

Utility throttling
This feature regulates the performance impact of maintenance utilities so that they can run concurrently during production periods. Although the *impact policy* for throttled utilities is defined by default, you must set the *impact priority* if you want to run a throttled utility. The throttling system ensures that the throttled utilities run as frequently as possible without violating the impact policy. Currently, you can throttle statistics collection, backup operations, rebalancing operations, and asynchronous index cleanup.

Automatic maintenance

The database manager provides automatic maintenance capabilities for performing database backups, keeping statistics current, and reorganizing tables and indexes as necessary. Performing maintenance activities on your databases is essential in ensuring that they are optimized for performance and recoverability.

Maintenance of your database includes some or all of the following activities:

- **Backups.** When you back up a database, the database manager takes a copy of the data in the database and stores it on a different medium in case of failure or damage to the original. Automatic database backups help to ensure that your database is backed up properly and regularly so that you don't have to worry about when to back up or know the syntax of the **BACKUP** command.
- **Data defragmentation (table or index reorganization).** This maintenance activity can increase the efficiency with which the database manager accesses your tables. Automatic reorganization manages an offline table and index reorganization so that you don't need to worry about when and how to reorganize your data.
- **Data access optimization (statistics collection).** The database manager updates the system catalog statistics on the data in a table, the data in indexes, or the data in both a table and its indexes. The optimizer uses these statistics to determine which path to use to access the data. Automatic statistics collection attempts to improve the performance of the database by maintaining up-to-date table statistics. The goal is to allow the optimizer to choose an access plan based on accurate statistics.
- **Statistics profiling.** Automatic statistics profiling advises when and how to collect table statistics by detecting outdated, missing, or incorrect statistics, and by generating statistical profiles based on query feedback.

It can be time-consuming to determine whether and when to run maintenance activities, but automatic maintenance removes the burden from you. You can manage the enablement of the automatic maintenance features simply and flexibly by using the automatic maintenance database configuration parameters. By setting the automatic maintenance database configuration parameters, you can specify your maintenance objectives The database manager uses these objectives to determine whether the maintenance activities need to be done and runs only the required ones during the next available maintenance window (a time period that you define).

In IBM Data Studio Version 3.1 or later, you can use the task assistant for configuring automatic maintenance. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Maintenance windows

A maintenance window is a time period that you define for the running of automatic maintenance activities, which are backups, statistics collection, statistics profiling, and reorganizations. An offline window might be the time period when access to a database is unavailable. An online window might be the time period when users are permitted to connect to a database.

A maintenance window is different from a task schedule. During a maintenance window, each automatic maintenance activity is not necessarily run. Instead, the database manager evaluates the system to determine the need for each maintenance activity to be run. If the maintenance requirements are not met, the maintenance activity is run. If the database is already well maintained, the maintenance activity is not run.

Think about when you want the automatic maintenance activities to be run. Automatic maintenance activities consume resources on your system and might affect the performance of your database when the activities are run. Some of these activities also restrict access to tables, indexes, and databases. Therefore, you must provide appropriate windows when the database manager can run maintenance activities.

Offline maintenance activities

Offline maintenance activities (offline database backups and table and index reorganizations) are maintenance activities that can occur only in the offline maintenance window. The extent to which user access is affected depends on which maintenance activity is running:

- During an offline backup, no applications can connect to the database. Any currently connected applications are forced off.
- During an offline table or index reorganization (data defragmentation), applications can access but not update the data in tables.

Offline maintenance activities run to completion even if they go beyond the window specified. Over time, the internal scheduling mechanism learns how to best estimate job completion times. If the offline maintenance window is too small for a particular database backup or reorganization activity, the scheduler will not start the job the next time and relies on the health monitor to provide notification of the need to increase the offline maintenance window.

Online maintenance activities

Online maintenance activities (automatic statistics collection and profiling, online index reorganizations, and online database backups) are maintenance activities that can occur only in the online maintenance window. When online maintenance activities run, any currently connected applications are allowed to remain connected, and new connections can be established. To minimize the impact on the system, online database backups and automatic statistics collection and profiling are throttled by the adaptive utility throttling mechanism.

Online maintenance activities run to completion even if they go beyond the window specified.

Self-tuning memory

A memory-tuning feature simplifies the task of memory configuration by automatically setting values for several memory configuration parameters. When enabled, the memory tuner dynamically distributes available memory resources among the following memory consumers: buffer pools, locking memory, package cache, and sort memory.

The tuner works within the memory limits that are defined by the **database_memory** configuration parameter. The value of this parameter can be automatically tuned as well. When self-tuning is enabled (when the value of **database_memory** has been set to AUTOMATIC), the tuner determines the overall memory requirements for the database and increases or decreases the amount of memory allocated for database shared memory, depending on current database requirements. For example, if current database requirements are high and there is sufficient free memory on the system, more memory is allocated for database shared memory. If the database memory requirements decrease, or if the amount of free memory on the system becomes too low, some database shared memory is released.

If the **database_memory** configuration parameter is not set to AUTOMATIC, the database uses the amount of memory that has been specified for this parameter, distributing it across the memory consumers as required. You can specify the amount of memory in one of two ways: by setting **database_memory** to some numeric value or by setting it to COMPUTED. In the latter case, the total amount of memory is based on the sum of the initial values of the database memory heaps at database startup time.

You can also enable the memory consumers for self tuning as follows:

- For buffer pools, use the ALTER BUFFERPOOL or the CREATE BUFFERPOOL statement (specifying the AUTOMATIC keyword)
- For locking memory, use the **locklist** or the **maxlocks** database configuration parameter (specifying a value of AUTOMATIC)
- For the package cache, use the **pckcachesz** database configuration parameter (specifying a value of AUTOMATIC)
- For sort memory, use the **sheapthres_shr** or the **sortheap** database configuration parameter (specifying a value of AUTOMATIC)

Changes resulting from self-tuning operations are recorded in memory tuning log files that are located in the stmmlog subdirectory. These log files contain summaries of the resource demands from each memory consumer during specific tuning intervals, which are determined by timestamps in the log entries.

If little memory is available, the performance benefits of self tuning will be limited. Because tuning decisions are based on database workload, workloads with rapidly changing memory requirements limit the effectiveness of the self-tuning memory manager (STMM). If the memory characteristics of your workload are constantly changing, the STMM will tune less frequently and under shifting target conditions. In this scenario, the STMM will not achieve absolute convergence, but will try instead to maintain a memory configuration that is tuned to the current workload.

Memory allocation

Memory allocation and deallocation occurs at various times. Memory might be allocated to a particular memory area when a specific event occurs (for example, when an application connects), or it might be reallocated in response to a configuration change.

Figure 3 on page 38 shows the different memory areas that the database manager allocates for various uses and the configuration parameters that enable you to control the size of these memory areas. Note that in a partitioned database environment, each database partition has its own database manager shared memory set.

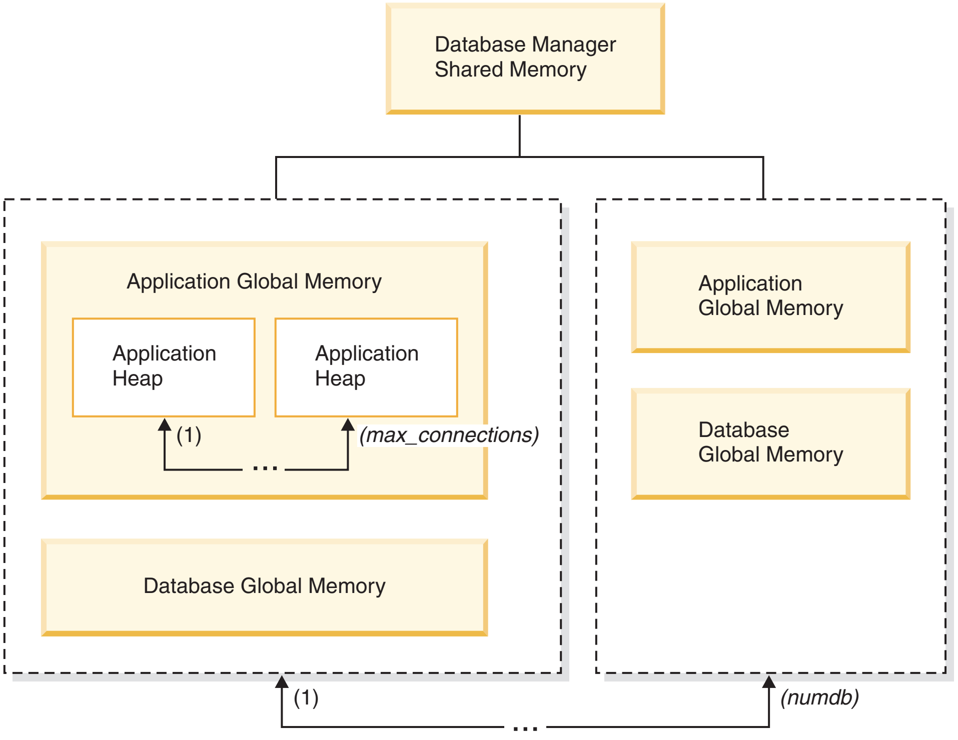


Figure 3. Types of memory allocated by the database manager

Memory is allocated by the database manager whenever one of the following events occurs:

When the database manager starts (db2start)

Database manager shared memory (also known as *instance shared memory*) remains allocated until the database manager stops (**db2stop**). This area contains information that the database manager uses to manage activity across all database connections. DB2 automatically controls the size of the database manager shared memory.

When a database is activated or connected to for the first time

Database global memory is used across all applications that connect to the database. The size of the database global memory is specified by the **database_memory** database configuration parameter. By default, this parameter is set to automatic, allowing DB2 to calculate the initial amount of memory allocated for the database and to automatically configure the database memory size during run time based on the needs of the database.

The following memory areas can be dynamically adjusted:

- Buffer pools (using the ALTER BUFFERPOOL statement)
- Database heap (including log buffers)
- Utility heap
- Package cache
- Catalog cache
- Lock list

The **sortheap**, **sheapthres_shr**, and **sheapthres** configuration parameters are also dynamically updatable. The only restriction is that **sheapthres** cannot be dynamically changed from 0 to a value that is greater than zero, or vice versa.

Shared sort operations are performed by default, and the amount of database shared memory that can be used by sort memory consumers at any one time is determined by the value of the **sheapthres_shr** database configuration parameter. Private sort operations are performed only if intra-partition parallelism, database partitioning, and the connection concentrator are all disabled, and the **sheapthres** database manager configuration parameter is set to a non-zero value.

When an application connects to a database

Each application has its own *application heap*, part of the *application global memory*. You can limit the amount of memory that any one application can allocate by using the **applheapsz** database configuration parameter, or limit overall application memory consumption by using the **appl_memory** database configuration parameter.

When an agent is created

Agent private memory is allocated for an agent when that agent is assigned as the result of a connect request or a new SQL request in a partitioned database environment. Agent private memory contains memory that is used only by this specific agent. If private sort operations have been enabled, the private sort heap is allocated from agent private memory.

The following configuration parameters limit the amount of memory that is allocated for each type of memory area. Note that in a partitioned database environment, this memory is allocated on each database partition.

numdb

This database manager configuration parameter specifies the maximum number of concurrent active databases that different applications can use. Because each database has its own global memory area, the amount of memory that can be allocated increases if you increase the value of this parameter.

maxappls

This database configuration parameter specifies the maximum number of applications that can simultaneously connect to a specific database. The value of this parameter affects the amount of memory that can be allocated for both agent private memory and application global memory for that database.

max_connections

This database manager configuration parameter limits the number of database connections or instance attachments that can access the data server at any one time.

max_coordagents

This database manager configuration parameter limits the number of database manager coordinating agents that can exist simultaneously across all active databases in an instance (and per database partition in a partitioned database environment). Together with **maxappls** and **max_connections**, this parameter limits the amount of memory that is allocated for agent private memory and application global memory.

You can use the memory tracker, invoked by the **db2mtrk** command, to view the current allocation of memory within the instance. You can also use the ADMIN_GET_MEM_USAGE table function to determine the total memory consumption for the entire instance or for just a single database partition. Use the MON_GET_MEMORY_SET and MON_GET_MEMORY_POOL table functions to examine the current memory usage at the instance, database, or application level.

On UNIX and Linux operating systems, although the **ipcs** command can be used to list all the shared memory segments, it does not accurately reflect the amount of resources consumed. You can use the **db2mtrk** command as an alternative to **ipcs**.

Self-tuning memory configuration

Enablement of self-tuning memory and memory consumers is controlled by database configuration parameters.

Self-tuning memory is enabled through the **self_tuning_mem** database configuration parameter.

The following memory-related database configuration parameters can be automatically tuned:

- **database_memory** - Database shared memory size
- **locklist** - Maximum storage for lock list
- **maxlocks** - Maximum percent of lock list before escalation
- **pckcachesz** - Package cache size
- **sheapthres_shr** - Sort heap threshold for shared sorts
- **sortheap** - Sort heap size

Enabling self-tuning memory

Self-tuning memory simplifies the task of memory configuration by automatically setting values for memory configuration parameters and sizing buffer pools.

About this task

When enabled, the memory tuner dynamically distributes available memory resources between several memory consumers, including buffer pools, locking memory, package cache, and sort memory.

Procedure

1. Enable self-tuning memory for the database by setting the **self_tuning_mem** database configuration parameter to ON using the **UPDATE DATABASE CONFIGURATION** command or the db2CfgSet API.
2. To enable the self tuning of memory areas that are controlled by memory configuration parameters, set the relevant configuration parameters to AUTOMATIC using the **UPDATE DATABASE CONFIGURATION** command or the db2CfgSet API.
3. To enable the self tuning of a buffer pool, set the buffer pool size to AUTOMATIC using the CREATE BUFFERPOOL statement or the ALTER BUFFERPOOL statement. In a partitioned database environment, that buffer pool should not have any entries in SYSCAT.BUFFERPOOLDBPARTITIONS.

Results

Note:

1. Because self-tuned memory is distributed between different memory consumers, at least two memory areas must be concurrently enabled for self tuning at any given time; for example, locking memory and database shared memory. The memory tuner actively tunes memory on the system (the value of the **self_tuning_mem** database configuration parameter is ON) when one of the following conditions is true:

- One configuration parameter or buffer pool size is set to AUTOMATIC, and the **database_memory** database configuration parameter is set to either a numeric value or to AUTOMATIC
- Any two of **locklist**, **sheapthres_shr**, **pckcachesz**, or buffer pool size is set to AUTOMATIC
- The **sortheap** database configuration parameter is set to AUTOMATIC

2. The value of the **locklist** database configuration parameter is tuned together with the **maxlocks** database configuration parameter. Disabling self tuning of the **locklist** parameter automatically disables self tuning of the **maxlocks** parameter, and enabling self tuning of the **locklist** parameter automatically enables self tuning of the **maxlocks** parameter.
3. Automatic tuning of **sortheap** or the **sheapthres_shr** database configuration parameter is allowed only when the database manager configuration parameter **sheapthres** is set to 0.
4. The value of **sortheap** is tuned together with **sheapthres_shr**. Disabling self tuning of the **sortheap** parameter automatically disables self tuning of the **sheapthres_shr** parameter, and enabling self tuning of the **sheapthres_shr** parameter automatically enables self tuning of the **sortheap** parameter.
5. Self-tuning memory runs only on the high availability disaster recovery (HADR) primary server. When self-tuning memory is activated on an HADR system, it will never run on the secondary server, and it runs on the primary server only if the configuration is set properly. If the HADR database roles are switched, self-tuning memory operations will also switch so that they run on the new primary server. After the primary database starts, or the standby database converts to a primary database through takeover, the self-tuning memory manager (STMM) engine dispatchable unit (EDU) might not start until the first client connects.

Disabling self-tuning memory

Self-tuning memory can be disabled for the entire database or for one or more configuration parameters or buffer pools.

About this task

If self-tuning memory is disabled for the entire database, the memory configuration parameters and buffer pools that are set to AUTOMATIC remain enabled for automatic tuning; however, the memory areas remain at their current size.

Procedure

1. Disable self-tuning memory for the database by setting the **self_tuning_mem** database configuration parameter to OFF using the **UPDATE DATABASE CONFIGURATION** command or the db2CfgSet API.
2. To disable the self tuning of memory areas that are controlled by memory configuration parameters, set the relevant configuration parameters to MANUAL or specify numeric parameter values using the **UPDATE DATABASE CONFIGURATION** command or the db2CfgSet API.
3. To disable the self tuning of a buffer pool, set the buffer pool size to a specific value using the ALTER BUFFERPOOL statement.

Results

Note:

- In some cases, a memory configuration parameter can be enabled for self tuning only if another related memory configuration parameter is also enabled. This means that, for example, disabling self-tuning memory for the **locklist** or the **sortheap** database configuration parameter disables self-tuning memory for the **maxlocks** or the **sheapthres_shr** database configuration parameter, respectively.

Determining which memory consumers are enabled for self tuning

You can view the self-tuning memory settings that are controlled by configuration parameters or that apply to buffer pools.

About this task

It is important to note that responsiveness of the memory tuner is limited by the time required to resize a memory consumer. For example, reducing the size of a buffer pool can be a lengthy process, and the performance benefits of trading buffer pool memory for sort memory might not be immediately realized.

Procedure

- To view the settings for configuration parameters, use one of the following methods:
  - Use the **GET DATABASE CONFIGURATION** command, specifying the **SHOW DETAIL** parameter.

    The memory consumers that can be enabled for self tuning are grouped together in the output as follows:

    ```
    Description                       Parameter         Current Value       Delayed Value
    --------------------------------------------------------------------------------------------
    Self tuning memory                    (SELF_TUNING_MEM) = ON (Active)        ON
    Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC(37200)   AUTOMATIC(37200)
    Max storage for lock list (4KB)              (LOCKLIST) = AUTOMATIC(7456)    AUTOMATIC(7456)
    Percent. of lock lists per application       (MAXLOCKS) = AUTOMATIC(98)      AUTOMATIC(98)
    Package cache size (4KB)                   (PCKCACHESZ) = AUTOMATIC(5600)    AUTOMATIC(5600)
    Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = AUTOMATIC(5000)    AUTOMATIC(5000)
    Sort list heap (4KB)                         (SORTHEAP) = AUTOMATIC(256)     AUTOMATIC(256)
    ```

  - Use the db2CfgGet API.

    The following values are returned:

    ```
    SQLF_OFF                0
    SQLF_ON_ACTIVE          2
    SQLF_ON_INACTIVE        3
    ```

    `SQLF_ON_ACTIVE` indicates that self tuning is both enabled and active, whereas `SQLF_ON_INACTIVE` indicates that self tuning is enabled but currently inactive.

- To view the self-tuning settings for buffer pools, use one of the following methods:
  - To retrieve a list of the buffer pools that are enabled for self tuning from the command line, use the following query:

    ```
    SELECT BPNAME, NPAGES FROM SYSCAT.BUFFERPOOLS
    ```

    When self tuning is enabled for a buffer pool, the NPAGES field in the SYSCAT.BUFFERPOOLS view for that particular buffer pool is set to -2. When self tuning is disabled, the NPAGES field is set to the current size of the buffer pool.

  - To determine the current size of buffer pools that are enabled for self tuning, use the **GET SNAPSHOT** command and examine the current size of the buffer pools (the value of the **bp_cur_buffsz** monitor element):

    ```
    GET SNAPSHOT FOR BUFFERPOOLS ON database-alias
    ```

An ALTER BUFFERPOOL statement that specifies the size of a buffer pool on a particular database partition creates an exception entry (or updates an existing entry) for that buffer pool in the SYSCAT.BUFFERPOOLDBPARTITIONS catalog view. If an exception entry for a buffer pool exists, that buffer pool does not participate in self-tuning operations when the default buffer pool size is set to AUTOMATIC.

Self-tuning memory in partitioned database environments

When using the self-tuning memory feature in partitioned database environments, there are a few factors that determine whether the feature will tune the system appropriately.

When self-tuning memory is enabled for partitioned databases, a single database partition is designated as the tuning partition, and all memory tuning decisions are based on the memory and workload characteristics of that database partition. After tuning decisions on that partition are made, the memory adjustments are distributed to the other database partitions to ensure that all database partitions maintain similar configurations.

The single tuning partition model assumes that the feature will be used only when all of the database partitions have similar memory requirements. Use the following guidelines when determining whether to enable self-tuning memory on your partitioned database.

Cases where self-tuning memory for partitioned databases is recommended

When all database partitions have similar memory requirements and are running on similar hardware, self-tuning memory can be enabled without any modifications. These types of environments share the following characteristics:

- All database partitions are on identical hardware, and there is an even distribution of multiple logical database partitions to multiple physical database partitions
- There is a perfect or near-perfect distribution of data
- Workloads are distributed evenly across database partitions, meaning that no database partition has higher memory requirements for one or more heaps than any of the others

In such an environment, if all database partitions are configured equally, self-tuning memory will properly configure the system.

Cases where self-tuning memory for partitioned databases is recommended with qualification

In cases where most of the database partitions in an environment have similar memory requirements and are running on similar hardware, it is possible to use self-tuning memory as long as some care is taken with the initial configuration. These systems might have one set of database partitions for data, and a much smaller set of coordinator partitions and catalog partitions. In such environments, it can be beneficial to configure the coordinator partitions and catalog partitions differently than the database partitions that contain data.

Self-tuning memory should be enabled on all of the database partitions that contain data, and one of these database partitions should be designated as the tuning partition. And because the coordinator and catalog partitions might be configured differently, self-tuning memory should be disabled on those partitions. To disable self-tuning memory on the coordinator and catalog partitions, set the **self_tuning_mem** database configuration parameter on these partitions to OFF.

Cases where self-tuning memory for partitioned databases is not recommended

If the memory requirements of each database partition are different, or if different database partitions are running on significantly different hardware, it is good practice to disable the self-tuning memory feature. You can disable the feature by setting the **self_tuning_mem** database configuration parameter to OFF on all partitions.

Comparing the memory requirements of different database partitions

The best way to determine whether the memory requirements of different database partitions are sufficiently similar is to consult the snapshot monitor. If the following snapshot elements are similar on all database partitions (differing by no more than 20%), the memory requirements of the database partitions can be considered sufficiently similar.

Collect the following data by issuing the command: `get snapshot for database on <dbname>`

```
Locks held currently                    = 0
Lock waits                              = 0
Time database waited on locks (ms)      = 0
Lock list memory in use (Bytes)         = 4968
Lock escalations                        = 0
Exclusive lock escalations              = 0

Total Shared Sort heap allocated        = 0
Shared Sort heap high water mark        = 0
Post threshold sorts (shared memory)    = 0
Sort overflows                          = 0

Package cache lookups                   = 13
Package cache inserts                   = 1
Package cache overflows                 = 0
Package cache high water mark (Bytes)   = 655360

Number of hash joins                    = 0
Number of hash loops                    = 0
Number of hash join overflows           = 0
Number of small hash join overflows     = 0
Post threshold hash joins (shared memory)  = 0

Number of OLAP functions                = 0
Number of OLAP function overflows       = 0
Active OLAP functions                   = 0
```

Collect the following data by issuing the command: `get snapshot for bufferpools on <dbname>`

```
Buffer pool data logical reads          = 0
Buffer pool data physical reads         = 0
Buffer pool index logical reads         = 0
Buffer pool index physical reads        = 0
Total buffer pool read time (milliseconds) = 0
Total buffer pool write time (milliseconds)= 0
```

Using self-tuning memory in partitioned database environments

When self-tuning memory is enabled in partitioned database environments, there is a single database partition (known as the *tuning partition*) that monitors the memory configuration and propagates any configuration changes to all other database partitions to maintain a consistent configuration across all the participating database partitions.

The tuning partition is selected on the basis of several characteristics, such as the number of database partitions in the partition group and the number of buffer pools.

- To determine which database partition is currently specified as the tuning partition, call the **ADMIN_CMD** procedure as follows:

  ```
  CALL SYSPROC.ADMIN_CMD('get stmm tuning dbpartitionnum')
  ```

- To change the tuning partition, call the **ADMIN_CMD** procedure as follows:

  ```
  CALL SYSPROC.ADMIN_CMD('update stmm tuning dbpartitionnum <partitionnum>')
  ```

  The tuning partition is updated asynchronously or at the next database startup. To have the memory tuner automatically select the tuning partition, enter -1 for the *partitionnum* value.

Starting the memory tuner in partitioned database environments

In a partitioned database environment, the memory tuner will start only if the database is activated by an explicit **ACTIVATE DATABASE** command, because self-tuning memory requires that all partitions be active.

Disabling self-tuning memory for a specific database partition

- To disable self-tuning memory for a subset of database partitions, set the **self_tuning_mem** database configuration parameter to OFF for those database partitions.
- To disable self-tuning memory for a subset of the memory consumers that are controlled by configuration parameters on a specific database partition, set the value of the relevant configuration parameter or the buffer pool size to MANUAL or to some specific value on that database partition. It is recommended that self-tuning memory configuration parameter values be consistent across all running partitions.
- To disable self-tuning memory for a particular buffer pool on a specific database partition, issue the ALTER BUFFERPOOL statement, specifying a size value and the partition on which self-tuning memory is to be disabled.

  An ALTER BUFFERPOOL statement that specifies the size of a buffer pool on a particular database partition will create an exception entry (or update an existing entry) for that buffer pool in the SYSCAT.BUFFERPOOLDBPARTITIONS catalog view. If an exception entry for a buffer pool exists, that buffer pool will not participate in self-tuning operations when the default buffer pool size is set to AUTOMATIC. To remove an exception entry so that a buffer pool can be enabled for self tuning:

  1. Disable self tuning for this buffer pool by issuing an ALTER BUFFERPOOL statement, setting the buffer pool size to a specific value.
  2. Issue another ALTER BUFFERPOOL statement to set the size of the buffer pool on this database partition to the default.
  3. Enable self tuning for this buffer pool by issuing another ALTER BUFFERPOOL statement, setting the buffer pool size to AUTOMATIC.

Enabling self-tuning memory in nonuniform environments

Ideally, data should be distributed evenly across all database partitions, and the workload that is run on each partition should have similar memory requirements. If the data distribution is skewed, so that one or more of your database partitions contain significantly more or less data than other database partitions, these anomalous database partitions should not be enabled for self tuning. The same is true if the memory requirements are skewed across the database partitions, which can happen, for example, if resource-intensive sorts are only performed on one partition, or if some database partitions are associated with different hardware and more available memory than others. Self tuning memory can still be enabled on some database partitions in this type of environment. To take advantage of self-tuning memory in environments with skew, identify a set of database partitions that have similar data and memory requirements and enable them for self tuning. Memory in the remaining partitions should be configured manually.

Configuring memory and memory heaps

With the simplified memory configuration feature, you can configure memory and memory heaps required by the DB2 data server by using the default AUTOMATIC setting for most memory-related configuration parameters, thereby, requiring much less tuning.

The simplified memory configuration feature provides the following benefits:

- You can use a single parameter, **instance_memory**, to specify all of the memory that the database manager is allowed to allocate from its private and shared memory heaps. Also, you can use the **appl_memory** configuration parameter to control the maximum amount of application memory that is allocated by DB2 database agents to service application requests.
- You are not required to manually tune parameters used solely for functional memory.
- You can use the **db2mtrk** command to monitor heap usage and the ADMIN_GET_MEM_USAGE table function to query overall memory consumption.
- The default DB2 configuration requires much less tuning, a benefit for new instances that you create.

The following table lists the memory configuration parameters whose values default to the AUTOMATIC setting. These parameters can also be configured dynamically, if necessary. Note that the meaning of the AUTOMATIC setting differs with each parameter, as described in the rightmost column.

Table 3. Memory configuration parameters whose values default to AUTOMATIC

Configuration parameter name	Description	Meaning of the AUTOMATIC setting
appl_memory	Controls the maximum amount of application memory that is allocated by DB2 database agents to service application requests.	If an **instance_memory** limit is enforced, the AUTOMATIC setting allows all application memory requests as long as the total amount of memory allocated by the database partition is within the **instance_memory** limit. Otherwise, it allows request as long as there are system resources available.

Table 3. Memory configuration parameters whose values default to AUTOMATIC (continued)

Configuration parameter name	Description	Meaning of the AUTOMATIC setting
applheapsz	Starting with Version 9.5, this parameter refers to the total amount of application memory that can be consumed by the entire application. For partitioned database environments, Concentrator, or SMP configurations, this means that you might need to increase the **applheapsz** value used in previous releases unless you use the AUTOMATIC setting.	The AUTOMATIC setting allows the application heap size to increase. as needed. A limit might be enforced if there is an **appl_memory** limit or an **instance_memory** limit.
database_memory	Specifies the amount of shared memory that is reserved for the database shared memory region.	When enabled, the memory tuner determines the overall memory requirements for the database and increases or decreases the amount of memory allocated for database shared memory depending on the current database requirements. Starting with Version 9.5, AUTOMATIC is the default setting for all DB2 server products.
dbheap	Determines the maximum memory used by the database heap.	The AUTOMATIC setting allows the database heap to increase as needed. A limit might be enforced if there is a **database_memory** limit or an **instance_memory** limit.
instance_memory	If you are using a DB2 database products with memory usage restrictions or if you set this parameter to a specific value, this parameter specifies the maximum amount of memory that can be allocated for a database partition.	The AUTOMATIC setting allows the overall memory consumed by the entire database manager instance to grow as needed, and STMM ensures that sufficient system memory is available to prevent memory overcommitment. For DB2 database products with memory usage restrictions, the AUTOMATIC setting enforces a limit based on the lower of a computed value (75-95% of RAM) and the allowable memory usage under the license. See **instance_memory** for details on when it is enforced as a limit.
mon_heap_sz	Determines the amount of the memory, in pages, to allocate for database system monitor data.	The AUTOMATIC setting allows the monitor heap to increase as needed. A limit might be enforced if there is an **instance_memory** limit.
stat_heap_sz	Indicates the maximum size of the heap used in collecting statistics using the **RUNSTATS** command.	The AUTOMATIC setting allows the statistics heap size to increase as needed. A limit might be enforced if there is an **appl_memory** limit or an **instance_memory** limit.

Table 3. Memory configuration parameters whose values default to AUTOMATIC (continued)

Configuration parameter name	Description	Meaning of the AUTOMATIC setting
stmtheap	Specifies the size of the statement heap which is used as a work space for the SQL or XQuery compiler to compile an SQL or XQuery statement.	The AUTOMATIC setting allows the statement heap to increase as needed. A limit might be enforced if there is an **appl_memory** limit or an **instance_memory** limit.

Note: The DBMCFG and DBCFG administrative views retrieve database manager configuration parameter information for the currently connected database for all database partitions. For the **mon_heap_sz**, **stmtheap**, and **stat_heap_sz** configuration parameters, the DEFERRED_VALUE column on this view does not persist across database activations. That is, when you issue the **get dbm cfg show detail** or **get db cfg show detail** command, the output from the query shows updated (in memory) values.

The following table shows whether configuration parameters are set to the default AUTOMATIC value during instance upgrade or creation and during database upgrade or creation.

Table 4. Configuration parameters set to AUTOMATIC during instance and database upgrade and creation

Configuration parameters	Set to AUTOMATIC upon instance upgrade or creation	Set to AUTOMATIC upon database upgrade	Set to AUTOMATIC upon database creation
applheapsz[1]		X	X
dbheap		X	X
instance_memory	X		
mon_heap_sz[1]	X		
stat_heap_sz[1]		X	X
stmtheap[1]			X

As part of the move to simplified memory configuration, the following elements have been deprecated:

- Configuration parameters **appgroup_mem_sz**, **groupheap_ratio**, and **app_ctl_heap_sz**. These configuration parameters are replaced with the new **appl_memory** configuration parameter.
- The **-p** parameter of the **db2mtrk** memory tracker command. This option, which lists private agent memory heaps, is replaced with the **-a** parameter, which lists all application memory consumption.

Agent and process model configuration

Starting with Version 9.5, DB2 databases feature a less complex and more flexible mechanism for configuring process model-related parameters. This simplified configuration eliminates the need for regular adjustments to these parameters and reduces the time and effort required to configure them. It also eliminates the need to shut down and restart DB2 instances to have the new values take effect.

To allow for dynamic and automatic agent and memory configuration, slightly more memory resources are required when an instance is activated.

Automatic storage

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

Databases use automatic storage by default

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

Note: Although, you can create a database specifying the AUTOMATIC STORAGE NO clause, the AUTOMATIC STORAGE clause is deprecated and might be removed from a future release.
By default, all databases are created with automatic storage. However, if the database is created specifying the AUTOMATIC STORAGE NO clause it cannot use automatic storage managed table spaces.

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

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

Important:

- Adding storage paths does not convert existing non-automatic storage table spaces to use automatic storage. You can convert database managed (DMS) table spaces to use automatic storage. System managed (SMS) table spaces cannot be converted to automatic storage. See “Converting table spaces to use automatic storage” on page 133 for more information.
- Once a database has storage groups created, it always has at least one storage group. You cannot remove the last storage group from the database manger.
- To help ensure predictable performance, the storage paths added to a storage group should have similar media characteristics.

Data compression

You can reduce storage needed for your data by using the compression capabilities built into DB2 for Linux, UNIX, and Windows to reduce the size of tables, indexes and even your backup images.

Tables and indexes often contain repeated information. This repetition can range from individual or combined column values, to common prefixes for column values, or to repeating patterns in XML data. There are a number of compression capabilities that you can use to reduce the amount of space required to store your tables and indexes, along with features you can employ to determine the savings compression can offer.

You can also use backup compression to reduce the size of your backups. [1]

Compression capabilities included with most editions of DB2 V9.7 include:

- Value compression
- Backup compression.

The following additional compression capabilities are available with the a license for the DB2 Storage Optimization Feature:

- Row compression, including compression for XML storage objects.
- Temporary table compression
- Index compression.

For more details about index compression, see “Index compression” on page 336.

For more details about backup compression, see “Backup compression” on page 740.

Automatic database backup

A database might become unusable due to a variety of hardware or software failures. Automatic database backup simplifies database backup management tasks for the DBA by always ensuring that a recent full backup of the database is performed as needed.

It determines the need to perform a backup operation based on one or more of the following measures:

- You have never completed a full database backup
- The time elapsed since the last full backup is more than a specified number of hours
- The transaction log space consumed since the last backup is more than a specified number of 4 KB pages (in archive logging mode only).

Protect your data by planning and implementing a disaster recovery strategy for your system. If suitable to your needs, you may incorporate the automatic database backup feature as part of your backup and recovery strategy.

If the database is enabled for roll-forward recovery (archive logging), then automatic database backup can be enabled for either online or offline backup. Otherwise, only offline backup is available. Automatic database backup supports disk, tape, Tivoli® Storage Manager (TSM), and vendor DLL media types.

If backup to disk is selected, the automatic backup feature will regularly delete backup images from the directory specified in the automatic database backup configuration. Only the most recent backup image will be available at any given time, regardless of the number of full backups that are specified in the automatic backup policy file. It is recommended that this directory be kept exclusively for the automatic backup feature and not be used to store other backup images.

[1] See “Backup compression” in *Data Recovery and High Availability Guide and Reference* for more information.

The automatic database backup feature can be enabled or disabled by using the **auto_db_backup** and **auto_maint** database configuration parameters. In a partitioned database environment, the automatic database backup runs on each database partition if the database configuration parameters are enabled on that database partition.

You can also configure automatic backup using one of the system stored procedures called AUTOMAINT_SET_POLICY and AUTOMAINT_SET_POLICYFILE.

Automatic table and index maintenance

After many changes to table data, a table and its indexes can become fragmented. Logically sequential data might be found on nonsequential pages, forcing additional read operations by the database manager to access data.

The statistical information that is collected by the **RUNSTATS** utility shows the distribution of data within a table. Analysis of these statistics can indicate when and what type of reorganization is necessary.

The automatic reorganization process determines the need for table or index reorganization by using formulas that are part of the **REORGCHK** utility. It periodically evaluates tables and indexes that had their statistics updated to see whether reorganization is required, and schedules such operations whenever they are necessary.

The automatic reorganization feature can be enabled or disabled through the **auto_reorg**, **auto_tbl_maint**, and **auto_maint** database configuration parameters.

In a partitioned database environment, the initiation of automatic reorganization is done on the catalog database partition. These configuration parameters are enabled only on the catalog database partition. The **REORG** operation, however, runs on all of the database partitions on which the target tables are found.

If you are unsure about when and how to reorganize your tables and indexes, you can incorporate automatic reorganization as part of your overall database maintenance plan.

You can also reorganize multidimensional clustering (MDC) and insert time clustering (ITC) tables to reclaim space. The freeing of extents from MDC and ITC tables is only supported for tables in DMS table spaces and automatic storage. Freeing extents from your MDC and ITC tables can be done in an online fashion with the RECLAIM EXTENTS option of the REORG TABLE command.

You can also schedule an alternate means to reclaim space from your indexes. The REORG INDEX command has an index clause in which you can specify space-reclaim-options. When you specify RECLAIM EXTENTS in space-reclaim-options, space is released back to the table space in an online fashion. This operation provides space reclamation without the need for a full rebuild of the indexes. The REBUILD option of the REORG INDEX command also reclaims space, but not necessarily in an online fashion.

Automatic reorganization on data partitioned tables

For DB2 Version 9.7 Fix Pack 1 and earlier releases, automatic reorganization supports reorganization of a data partitioned table for the entire table. For DB2

V9.7 Fix Pack 1 and later releases, automatic reorganization supports reorganizing data partitions of a partitioned table and reorganizing the partitioned indexes on a data partition of a partitioned table.

To avoid placing an entire data partitioned table into ALLOW NO ACCESS mode, automatic reorganization performs **REORG INDEXES ALL** operations at the data partition level on partitioned indexes that need to be reorganized. Automatic reorganization performs **REORG INDEX** operations on any nonpartitioned index that needs to be reorganized.

Automatic reorganization performs the following **REORG TABLE** operations on data partitioned tables:

- If any nonpartitioned indexes (except system-generated XML path indexes) are defined on the table and there is only one partition that needs to be reorganized, automatic reorganization performs a **REORG TABLE** operation with the **ON DATA PARTITION** clause to specify the partition that needs to be reorganized. Otherwise, automatic reorganization performs a **REORG TABLE** on the entire table without the **ON DATA PARTITION** clause.
- If no nonpartitioned indexes (except system-generated XML path indexes) are defined on the table, automatic reorganization performs a **REORG TABLE** operation with the **ON DATA PARTITION** clause on each partition that needs to be reorganized.

Automatic reorganization on volatile tables

You can enable automatic index reorganization for volatile tables. The automatic reorganization process determines whether index reorganization is required for volatile tables and schedules a REORG INDEX CLEANUP. Index reorganization is performed periodically on volatile tables and releases space that can be reused by the indexes defined on these tables.

Statistics cannot be collected in volatile tables because they are updated frequently. To determine what indexes need to be reorganized, automatic reorganization uses the numInxPseudoEmptyPagesForVolatile attribute instead of REORGCHK. The number of pseudo empty pages is maintained internally, visible through mon_get_index, and does not require a RUNSTATS operation like REORGCHK. This attribute in the AUTO_REORG policy indicates how many empty index pages with pseudo deleted keys an index must have so index reorganization is triggered.

To enable automatic index reorganization in volatile tables:

- The **DB2_WORKLOAD** registry variable must be set to SAP.
- Automatic reorganization must be enabled.
- The numInxPseudoEmptyPagesForVolatile attribute must be set.

Automatic statistics collection

The DB2 optimizer uses catalog statistics to determine the most efficient access plan for a query. Out-of-date or incomplete table or index statistics might lead the optimizer to select a suboptimal plan, which slows down query execution. However, deciding which statistics to collect for a given workload is complex, and keeping these statistics up-to-date is time-consuming.

With automatic statistics collection, part of the DB2 automated table maintenance feature, you can let the database manager determine whether statistics need to be updated. Automatic statistics collection can occur *synchronously* at statement compilation time by using the real-time statistics (RTS) feature, or the **RUNSTATS** command can be enabled to simply run in the background for *asynchronous* collection. Although background statistics collection can be enabled while real-time statistics collection is disabled, background statistics collection must be enabled for real-time statistics collection to occur. Automatic background statistics collection **auto_runstats** and automatic real-time statistics collection **auto_stmt_stats** are enabled by default when you create a database.

Starting with DB2 Version 9, you can use the Configuration Advisor to determine the initial configuration for new databases including the appropriate setting for the **auto_stmt_stats** database configuration parameter.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for configuring automatic statistics collection. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Understanding asynchronous and real-time statistics collection

When real-time statistics collection is enabled, statistics can be fabricated by using certain metadata. *Fabrication* means deriving or creating statistics, rather than collecting them as part of normal **RUNSTATS** command activity. For example, the number of rows in a table can be derived from knowing the number of pages in the table, the page size, and the average row width. In some cases, statistics are not derived, but are maintained by the index and data manager and can be stored directly in the catalog. For example, the index manager maintains a count of the number of leaf pages and levels in each index.

The query optimizer determines how statistics are collected, based on the needs of the query and the amount of table update activity (the number of update, insert, or delete operations).

Real-time statistics collection provides more timely and more accurate statistics. Accurate statistics can result in better query execution plans and improved performance. Regardless of whether real-time statistics is enabled, asynchronous statistics collection occurs at two-hour intervals. This interval might not be frequent enough to provide accurate statistics for some applications.

Real-time statistics collection also initiates asynchronous collection requests when:

- Table activity is not high enough to require synchronous collection, but is high enough to require asynchronous collection
- Synchronous statistics collection used sampling because the table was large
- Synchronous statistics were fabricated
- Synchronous statistics collection failed because the collection time was exceeded

At most, two asynchronous requests can be processed at the same time, but only for different tables. One request must have been initiated by real-time statistics collection, and the other must have been initiated by asynchronous statistics collection checking.

The performance impact of automatic statistics collection is minimized in several ways:

- Asynchronous statistics collection is performed by using a throttled **RUNSTATS** utility. Throttling controls the amount of resource that is consumed by the

RUNSTATS utility, based on current database activity: as database activity increases, the utility runs more slowly, reducing its resource demands.

- Synchronous statistics collection is limited to 5 seconds per query. This value can be controlled by the RTS optimization guideline. If synchronous collection exceeds the time limit, an asynchronous collection request is submitted.
- Synchronous statistics collection does not store the statistics in the system catalog. Instead, the statistics are stored in a statistics cache and are later stored in the system catalog by an asynchronous operation. This storage sequence avoids the overhead and possible lock contention involved when updating the system catalog. Statistics in the statistics cache are available for subsequent SQL compilation requests.
- Only one synchronous statistics collection operation occurs per table. Other agents requiring synchronous statistics collection fabricate statistics, if possible, and continue with statement compilation. This behavior is also enforced in a partitioned database environment, where agents on different database partitions might require synchronous statistics.
- You can customize the type of statistics that are collected by enabling statistics profiling, which uses information about previous database activity to determine which statistics are required by the database workload, or by creating your own statistics profile for a particular table.
- Only tables with missing statistics or high levels of activity (as measured by the number of update, insert, or delete operations) are considered for statistics collection. Even if a table meets the statistics collection criteria, synchronous statistics are not collected unless query optimization requires them. In some cases, the query optimizer can choose an access plan without statistics.
- For asynchronous statistics collection checking, large tables (tables with more than 4000 pages) are sampled to determine whether high table activity changed the statistics. Statistics for such large tables are collected only if warranted.
- For asynchronous statistics collection, the **RUNSTATS** utility is automatically scheduled to run during the online maintenance window that is specified in your maintenance policy. This policy also specifies the set of tables that are within the scope of automatic statistics collection, further minimizing unnecessary resource consumption.
- Synchronous statistics collection and fabrication do not follow the online maintenance window that is specified in your maintenance policy, because synchronous requests must occur immediately and have limited collection time. Synchronous statistics collection and fabrication follow the policy that specifies the set of tables that are within the scope of automatic statistics collection.
- While automatic statistics collection is being performed, the affected tables are still available for regular database activity (update, insert, or delete operations).
- Real-time statistics (synchronous or fabricated) are not collected for nicknames. To refresh nickname statistics in the system catalog for synchronous statistics collection, call the SYSPROC.NNSTAT procedure. For asynchronous statistics collection, DB2 for Linux, UNIX, and Windows automatically calls the SYSPROC.NNSAT procedure to refresh the nickname statistics in the system catalog.
- Real-time statistics (synchronous or fabricated) are not collected for statistical views.
- Declared temporary tables (DGTTs) can have only Real-time statistics collected.

Although real-time statistics collection is designed to minimize statistics collection overhead, try it in a test environment first to ensure that there is no negative performance impact. There might be a negative performance impact in some online transaction processing (OLTP) scenarios, especially if there is an upper boundary for how long a query can run.

Real-time synchronous statistics collection is performed for regular tables, materialized query tables (MQTs), and global temporary tables. Asynchronous statistics are not collected for global temporary tables. Global temporary tables cannot be excluded from real-time statistics via the automatic maintenance policy facility.

Automatic statistics collection (synchronous or asynchronous) does not occur for:

- Tables that are marked VOLATILE (tables that have the VOLATILE field set in the SYSCAT.TABLES catalog view)
- Created temporary tables (CGTTs)
- Tables that had their statistics manually updated, by issuing UPDATE statements directly against SYSSTAT catalog views

  When you modify table statistics manually, the database manager assumes that you are now responsible for maintaining their statistics. To induce the database manager to maintain statistics for a table that had its statistics manually updated, collect statistics by using the **RUNSTATS** command or specify statistics collection when using the **LOAD** command. Tables created before Version 9.5 that had their statistics updated manually before upgrading are not affected, and their statistics are automatically maintained by the database manager until they are manually updated.

Statistics fabrication does not occur for:

- Statistical views
- Tables that had their statistics manually updated, by issuing UPDATE statements directly against SYSSTAT catalog views. If real-time statistics collection is not enabled, some statistics fabrication still occurs for tables that had their statistics manually updated.

In a partitioned database environment, statistics are collected on a single database partition and then extrapolated. The database manager always collects statistics (both synchronous and asynchronous) on the first database partition of the database partition group.

No real-time statistics collection activity will occur until at least five minutes after database activation.

Real-time statistics processing occurs for both static and dynamic SQL.

A table that was truncated, either by using the TRUNCATE statement or by using the **IMPORT** command, is automatically recognized as having out of date statistics.

Automatic statistics collection, both synchronous and asynchronous, invalidates cached dynamic statements that reference tables for which statistics were collected. This is done so that cached dynamic statements can be re-optimized with the latest statistics.

Asynchronous automatic statistics collection operations might be interrupted when the database is deactivated. If the database was not explicitly activated using the **ACTIVATE DATABASE** command or API, then the database is deactivated when the last user disconnects from the database. If operations are interrupted, then error messages might be recorded in the DB2 diagnostic log file. To avoid interrupting asynchronous automatic statistics collection operations, explicitly activate the database.

Real-time statistics and explain processing

There is no real-time processing for a query that is only explained (not executed) by using the EXPLAIN facility. The following table summarizes the behavior under different values of the CURRENT EXPLAIN MODE special register.

Table 5. Real-time statistics collection as a function of the value of the CURRENT EXPLAIN MODE special register

CURRENT EXPLAIN MODE value	Real-time statistics collection considered
YES	Yes
EXPLAIN	No
NO	Yes
REOPT	Yes
RECOMMEND INDEXES	No
EVALUATE INDEXES	No

Automatic statistics collection and the statistics cache

A statistics cache was introduced in DB2 Version 9.5 to make synchronously collected statistics available to all queries. This cache is part of the catalog cache. In a partitioned database environment, the statistics cache resides only on the catalog database partition even though each database partition has a catalog cache. When real-time statistics collection is enabled, catalog cache requirements are higher. Consider tuning the value of the **catalogcache_sz** database configuration parameter when real-time statistics collection is enabled.

Automatic statistics collection and statistical profiles

Synchronous and asynchronous statistics are collected according to a statistical profile that is in effect for a table, with the following exceptions:

- To minimize the overhead of synchronous statistics collection, the database manager might collect statistics by using sampling. In this case, the sampling rate and method might be different from those rates and methods that are specified in the statistical profile.
- Synchronous statistics collection might choose to fabricate statistics, but it might not be possible to fabricate all statistics that are specified in the statistical profile. For example, column statistics such as COLCARD, HIGH2KEY, and LOW2KEY cannot be fabricated unless the column is leading in some index.

If synchronous statistics collection cannot collect all statistics that are specified in the statistical profile, an asynchronous collection request is submitted.

The following sections explain different operating characteristics of automatic statistics collection.

Configuration Advisor

You can use the Configuration Advisor to obtain recommendations for the initial values of the buffer pool size, database configuration parameters, and database manager configuration parameters.

To use the Configuration Advisor, specify the **AUTOCONFIGURE** command for an existing database, or specify **AUTOCONFIGURE** as an option of the **CREATE DATABASE** command. To configure your database, you must have SYSADM, SYSCTRL, or SYSMAINT authority.

You can display the recommended values or apply them by specifying the APPLY parameter in the **CREATE DATABASE** and **AUTOCONFIGURE** commands. The recommendations are based on input that you provide and system information that the advisor gathers.

The values suggested by the Configuration Advisor are relevant for only one database per instance. If you want to use this advisor on more than one database, each database must belong to a separate instance.

Tuning configuration parameters using the Configuration Advisor

The Configuration Advisor helps you to tune performance and to balance memory requirements for a single database per instance by suggesting which configuration parameters to modify and suggesting values for them. The Configuration Advisor is automatically run when you create a database.

About this task

To disable this feature or to explicitly enable it, use the **db2set** command before creating a database, as follows:

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

To define values for several of the configuration parameters and to determine the scope of the application of those parameters, use the **AUTOCONFIGURE** command, specifying one of the following options:

- NONE, meaning that none of the values are applied
- DB ONLY, meaning that only database configuration and buffer pool values are applied
- DB AND DBM, meaning that all parameters and their values are applied

Note: Even if you automatically enabled the Configuration Advisor when you ran the **CREATE DATABASE** command, you can still specify **AUTOCONFIGURE** command options. If you did not enable the Configuration Advisor when you ran the **CREATE DATABASE** command, you can run the Configuration Advisor manually afterwards.

Example: Requesting configuration recommendations using the Configuration Advisor

This scenario demonstrates to run the Configuration Advisor from the command line to generate recommendations and shows the output that the Configuration Advisor produces.

To run the Configuration Advisor:

1. Connect to the PERSONL database by specifying the following command from the command line:

   ```
   DB2 CONNECT TO PERSONL
   ```

2. Issue the **AUTOCONFIGURE** command from the CLP, specifying how the database is used. As shown in the following example, set a value of NONE for the **APPLY** option to indicate that you want to view the configuration recommendations but not apply them:

   ```
      DB2 AUTOCONFIGURE USING
           MEM_PERCENT 60
           WORKLOAD_TYPE MIXED
           NUM_STMTS 500
           ADMIN_PRIORITY BOTH
           IS_POPULATED YES
           NUM_LOCAL_APPS 0
           NUM_REMOTE_APPS 20
           ISOLATION RR
           BP_RESIZEABLE YES
      APPLY NONE
   ```

   If you are unsure about the value of a parameter for the command, you can omit it, and the default will be used. You can pass up to 10 parameters without values: MEM_PERCENT, WORKLOAD_TYPE, and so on, as shown in the previous example.

The recommendations generated by the **AUTOCONFIGURE** command are displayed on the screen in table format, as shown in Figure 4 on page 59

```
       Former and Applied Values for Database Manager Configuration
Description                              Parameter   Current Value   Recommended Value
-----------------------------------------------------------------------------------------
Application support layer heap size (4KB)   (ASLHEAPSZ) = 15             15
No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = AUTOMATIC      AUTOMATIC
Enable intra-partition parallelism     (INTRA_PARALLEL) = NO             NO
Maximum query degree of parallelism   (MAX_QUERYDEGREE) = ANY            1
Agent pool size                        (NUM_POOLAGENTS) = 100(calculated) 200
Initial number of agents in pool       (NUM_INITAGENTS) = 0              0
Max requester I/O block size (bytes)         (RQRIOBLK) = 32767          32767
Sort heap threshold (4KB)                  (SHEAPTHRES) = 0              0

       Former and Applied Values for Database Configuration
Description                              Parameter   Current Value   Recommended Value
-----------------------------------------------------------------------------------------
Default application heap (4KB)             (APPLHEAPSZ) = 256            256
Catalog cache size (4KB)              (CATALOGCACHE_SZ) = (MAXAPPLS*4)   260
Changed pages threshold                (CHNGPGS_THRESH) = 60             80
Database heap (4KB)                            (DBHEAP) = 1200           2791
Degree of parallelism                      (DFT_DEGREE) = 1              1
Default tablespace extentsize (pages)   (DFT_EXTENT_SZ) = 32             32
Default prefetch size (pages)         (DFT_PREFETCH_SZ) = AUTOMATIC      AUTOMATIC
Default query optimization class         (DFT_QUERYOPT) = 5              5
Max storage for lock list (4KB)              (LOCKLIST) = 100            AUTOMATIC
Log buffer size (4KB)                        (LOGBUFSZ) = 8              99
Log file size (4KB)                         (LOGFILSIZ) = 1000           1024
Number of primary log files                (LOGPRIMARY) = 3              8
Number of secondary log files               (LOGSECOND) = 2              3
Max number of active applications            (MAXAPPLS) = AUTOMATIC      AUTOMATIC
Percent. of lock lists per application       (MAXLOCKS) = 10             AUTOMATIC
Group commit count                          (MINCOMMIT) = 1              1
Number of asynchronous page cleaners   (NUM_IOCLEANERS) = 1              1
Number of I/O servers                   (NUM_IOSERVERS) = 3              4
Package cache size (4KB)                   (PCKCACHESZ) = (MAXAPPLS*8)   1533
Percent log file reclaimed before soft chckpt (SOFTMAX) = 100            320
Sort list heap (4KB)                         (SORTHEAP) = 256            AUTOMATIC
statement heap (4KB)                         (STMTHEAP) = 4096           4096
Statistics heap size (4KB)               (STAT_HEAP_SZ) = 4384           4384
Utilities heap size (4KB)                (UTIL_HEAP_SZ) = 5000           113661
Self tuning memory                    (SELF_TUNING_MEM) = ON             ON
    Automatic runstats                  (AUTO_RUNSTATS) = ON             ON
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = 5000           AUTOMATIC

       Former and Applied Values for Bufferpool(s)
Description                              Parameter   Current Value   Recommended Value
-----------------------------------------------------------------------------------------
IBMDEFAULTBP                             Bufferpool size = -2             340985

DB210203I AUTOCONFIGURE completed successfully. Database manager or database
configuration values may have been changed. The instance must be restarted before
any changes come into effect. You may also want to rebind your packages after the
new configuration parameters take effect so that the new values will be used.
```

Figure 4. Configuration Advisor sample output

If you agree with all of the recommendations, either reissue the **AUTOCONFIGURE** command but specify that you want the recommended values to be applied by using the APPLY option, or update individual configuration parameters using the **UPDATE DATABASE MANAGER CONFIGURATION** command and the **UPDATE DATABASE CONFIGURATION** command.

Utility throttling

Utility throttling regulates the performance impact of maintenance utilities so that they can run concurrently during production periods. Although the impact policy, a setting that allows utilities to run in throttled mode, is defined by default, you must set the impact priority, a setting that each cleaner has indicating its throttling priority, when you run a utility if you want to throttle it.

The throttling system ensures that the throttled utilities are run as frequently as possible without violating the impact policy. You can throttle statistics collection, backup operations, rebalancing operations, and asynchronous index cleanups.

You define the impact policy by setting the **util_impact_lim** configuration parameter.

Cleaners are integrated with the utility throttling facility. By default, each (index) cleaner has a utility impact priority of 50 (acceptable values are between 1 and 100, with 0 indicating no throttling). You can change the priority by using the **SET UTIL_IMPACT_PRIORITY** command or the db2UtilityControl API.

Asynchronous index cleanup

Asynchronous index cleanup (AIC) is the deferred cleanup of indexes following operations that invalidate index entries. Depending on the type of index, the entries can be record identifiers (RIDs) or block identifiers (BIDs). Invalid index entries are removed by index cleaners, which operate asynchronously in the background.

AIC accelerates the process of detaching a data partition from a partitioned table, and is initiated if the partitioned table contains one or more nonpartitioned indexes. In this case, AIC removes all nonpartitioned index entries that refer to the detached data partition, and any pseudo-deleted entries. After all of the indexes have been cleaned, the identifier that is associated with the detached data partition is removed from the system catalog. In DB2 Version 9.7 Fix Pack 1 and later releases, AIC is initiated by an asynchronous partition detach task.

Prior to DB2 Version 9.7 Fix Pack 1, if the partitioned table has dependent materialized query tables (MQTs), AIC is not initiated until after a SET INTEGRITY statement is executed.

Normal table access is maintained while AIC is in progress. Queries accessing the indexes ignore any invalid entries that have not yet been cleaned.

In most cases, one cleaner is started for each nonpartitioned index that is associated with the partitioned table. An internal task distribution daemon is responsible for distributing the AIC tasks to the appropriate table partitions and assigning database agents. The distribution daemon and cleaner agents are internal system applications that appear in **LIST APPLICATIONS** command output with the application names **db2taskd** and **db2aic**, respectively. To prevent accidental disruption, system applications cannot be forced. The distribution daemon remains online as long as the database is active. The cleaners remain active until cleaning has been completed. If the database is deactivated while cleaning is in progress, AIC resumes when you reactivate the database.

AIC impact on performance

AIC incurs minimal performance impact.

An instantaneous row lock test is required to determine whether a pseudo-deleted entry has been committed. However, because the lock is never acquired, concurrency is unaffected.

Each cleaner acquires a minimal table space lock (IX) and a table lock (IS). These locks are released if a cleaner determines that other applications are waiting for locks. If this occurs, the cleaner suspends processing for 5 minutes.

Cleaners are integrated with the utility throttling facility. By default, each cleaner has a utility impact priority of 50. You can change the priority by using the **SET UTIL_IMPACT_PRIORITY** command or the db2UtilityControl API.

Monitoring AIC

You can monitor AIC with the **LIST UTILITIES** command. Each index cleaner appears as a separate utility in the output. The following is an example of output from the **LIST UTILITIES SHOW DETAIL** command:

```
ID                              = 2
Type                            = ASYNCHRONOUS INDEX CLEANUP
Database Name                   = WSDB
Partition Number                = 0
Description                     = Table: USER1.SALES, Index: USER1.I2
Start Time                      = 12/15/2005 11:15:01.967939
State                           = Executing
Invocation Type                  = Automatic
Throttling:
   Priority                      = 50
Progress Monitoring:
      Total Work                 = 5 pages
      Completed Work             = 0 pages
      Start Time                 = 12/15/2005 11:15:01.979033

ID                              = 1
Type                            = ASYNCHRONOUS INDEX CLEANUP
Database Name                   = WSDB
Partition Number                = 0
Description                     = Table: USER1.SALES, Index: USER1.I1
Start Time                      = 12/15/2005 11:15:01.978554
State                            = Executing
Invocation Type                  = Automatic
Throttling:
   Priority                      = 50
Progress Monitoring:
      Total Work                 = 5 pages
      Completed Work             = 0 pages
      Start Time                 = 12/15/2005 11:15:01.980524
```

In this case, there are two cleaners operating on the USERS1.SALES table. One cleaner is processing index I1, and the other is processing index I2. The progress monitoring section shows the estimated total number of index pages that need cleaning and the current number of clean index pages.

The `State` field indicates the current state of a cleaner. The normal state is Executing, but the cleaner might be in Waiting state if it is waiting to be assigned to an available database agent or if the cleaner is temporarily suspended because of lock contention.

Note that different tasks on different database partitions can have the same utility ID, because each database partition assigns IDs to tasks that are running on that database partition only.

Asynchronous index cleanup for MDC tables

You can enhance the performance of a rollout deletion-an efficient method for deleting qualifying blocks of data from multidimensional clustering (MDC) tables-by using asynchronous index cleanup (AIC). AIC is the deferred cleanup of indexes following operations that invalidate index entries.

Indexes are cleaned up synchronously during a standard rollout deletion. When a table contains many record ID (RID) indexes, a significant amount of time is spent removing the index keys that reference the table rows that are being deleted. You can speed up the rollout by specifying that these indexes are to be cleaned up after the deletion operation commits.

To take advantage of AIC for MDC tables, you must explicitly enable the *deferred index cleanup rollout* mechanism. There are two methods of specifying a deferred rollout: setting the **DB2_MDC_ROLLOUT** registry variable to DEFER or issuing the SET CURRENT MDC ROLLOUT MODE statement. During a deferred index cleanup rollout operation, blocks are marked as rolled out without an update to the RID indexes until after the transaction commits. Block identifier (BID) indexes are cleaned up during the delete operation because they do not require row-level processing.

AIC rollout is invoked when a rollout deletion commits or, if the database was shut down, when the table is first accessed following database restart. While AIC is in progress, queries against the indexes are successful, including those that access the index that is being cleaned up.

There is one coordinating cleaner per MDC table. Index cleanup for multiple rollouts is consolidated within the cleaner, which spawns a cleanup agent for each RID index. Cleanup agents update the RID indexes in parallel. Cleaners are also integrated with the utility throttling facility. By default, each cleaner has a utility impact priority of 50 (acceptable values are between 1 and 100, with 0 indicating no throttling). You can change this priority by using the **SET UTIL_IMPACT_PRIORITY** command or the db2UtilityControl API.

Note: In DB2 Version 9.7 and later releases, deferred cleanup rollout is not supported on a data partitioned MDC table with partitioned RID indexes. Only the NONE and IMMEDIATE modes are supported. The cleanup rollout type will be IMMEDIATE if the **DB2_MDC_ROLLOUT** registry variable is set to DEFER, or if the CURRENT MDC ROLLOUT MODE special register is set to DEFERRED to override the **DB2_MDC_ROLLOUT** setting.

If only nonpartitioned RID indexes exist on the MDC table, deferred index cleanup rollout is supported. The MDC block indexes can be partitioned or nonpartitioned.

Monitoring the progress of deferred index cleanup rollout operation

Because the rolled-out blocks on an MDC table are not reusable until after the cleanup is complete, it is useful to monitor the progress of a deferred index cleanup rollout operation. Use the **LIST UTILITIES** command to display a utility monitor entry for each index being cleaned up. You can also retrieve the total number of MDC table blocks in the database that are pending asynchronous cleanup following a rollout deletion (BLOCKS_PENDING_CLEANUP) by using the ADMIN_GET_TAB_INFO table function or the **GET SNAPSHOT** command.

In the following sample output for the **LIST UTILITIES SHOW DETAIL** command, progress is indicated by the number of pages in each index that have been cleaned up. Each phase represents one RID index.

```
ID                               = 2
Type                             = MDC ROLLOUT INDEX CLEANUP
Database Name                    = WSDB
Partition Number                 = 0
Description                      = TABLE.<schema_name>.<table_name>
Start Time                       = 06/12/2006 08:56:33.390158
State                            = Executing
Invocation Type                  = Automatic
Throttling:
   Priority                      = 50
Progress Monitoring:
   Estimated Percentage Complete = 83
   Phase Number                  = 1
      Description                = <schema_name>.<index_name>
      Total Work                 = 13 pages
      Completed Work             = 13 pages
      Start Time                 = 06/12/2006 08:56:33.391566
   Phase Number                  = 2
      Description                = <schema_name>.<index_name>
      Total Work                 = 13 pages
      Completed Work             = 13 pages
      Start Time                 = 06/12/2006 08:56:33.391577
   Phase Number                  = 3
      Description                = <schema_name>.<index_name>
      Total Work                 = 9 pages
      Completed Work             = 3 pages
      Start Time                 = 06/12/2006 08:56:33.391587
```

Chapter 4. IBM Data Studio

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

IBM Data Studio consist of the following components:

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

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

Related information:

IBM Data Studio documentation

Features in IBM Data Studio

IBM Data Studio product Web page

Download IBM Data Studio

Managing jobs in IBM Data Studio

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

With the Data Studio web console job manager you can:

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

  SQL-only scripts

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

DB2 CLP scripts

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

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

Executable/shell Scripts

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

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

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

Creating and managing jobs

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

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

Job List

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

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

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

Schedules

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

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

Notifications

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

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

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

History

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

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

Scenario: Creating and scheduling a job

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

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

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

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

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

First Alan verifies with Doug that the script has been tested and verified by development, and that it runs without problems on their test databases. Doug uses other IBM Data Studio tools to verify the scripts.

Next, Alan opens the Databases page in the web console to verify that the Sales database exists as a database connection. If needed, he adds a database connection to the Sales database with information from Becky, the owner of the Sales database. Becky wants to restrict the running of jobs on the **`Sales`** database to a specific subset of users, so Alan configures the database connection to connect with a user ID that has the minimum required authority of **`CONNECT`**. To schedule the job on the Sales database Alan also needs the user credentials of a user ID that has the authorizations on the database required by the actions that the script runs. That user ID also needs the required authority to run the cleanup job afterward.

Alan then opens the Job Manager page in the web console, and clicks **Add Job** in the Job List tab to create the job. After filling out the basic information, such as a job name and a description of the job, Alan selects the correct type of job to match the script and verifies that the job is enabled for scheduling.

Working through the new job wizard, Alan pastes in the script that Doug provided into the Script component of the job, making sure that the closing character defined for the job matches what is in the script.

Alan then creates a schedule from the Schedules component of the job, setting a date and time for the first job run, and configuring it to run monthly on the **`Sales`** database. As the user ID used in the database connection does not have the correct authority to run some of the commands in the script, Alan selects to run the job as the specific user ID with the correct authority that was provided by the database owner.

Alan also adds the requested cleanup job to the job in the Chain component. As the only required chained job is the cleanup, Alan adds it to run at the end of the chain.

Finally, Alan adds the email addresses of Doug and Doug's manager to the Notifications component of the job, and configures notifications to be sent if the job fails.

The job is now scheduled, and Alan can view the job, schedule, and notification information for the job in the corresponding job manager tabs. Once the job has been run, any user with access to the web console can use the History page to view the job history for the job, and get a detailed view by looking at the log entry for the job. If Doug does not have access to the web console, Alan adds Doug as a repository database user and uses the Console Security page to grant Doug access the web console.

Importing tasks from DB2 Task Center

Use the Data Studio web console to import existing tasks from the Task Center in the DB2 Control Center. Imported tasks are saved as jobs in the job manager.

About this task

The imported tasks are mapped to the appropriate job manager type as shown in the following table:

Table 6. Mapping of Task Center script type to Job Manager job type

Task Center script type	Job Manager job type
DB2 command script	DB2 CLP script
OS command script	Shell/Executable script

Restrictions: The following restrictions apply to importing tasks from the DB2 Task Center:

- Task types from DB2 Task Center:

Table 7. Restrictions for task types from DB2 Task Center

Task type	Restrictions
MVS shell script	Not supported.
Grouping	Not supported.
OS command script	The script interpreters and command execution parameters are not supported. The default script interpreter is used instead.
DB2 command script	Supported.

- Schedules that are associated with tasks from DB2 Task Center:

Table 8. Restrictions for schedules from DB2 Task Center

Schedule type	Restrictions
Weekly	Only schedules set for 1 to 4 weeks are supported.
Monthly	Only schedules set for 1 month and schedules set to a specific date or last date are supported.
Yearly	Only schedules set for 1 year are supported.
Expired (that is, schedules with a starting or ending time that is earlier than the current time)	Expired schedules will be imported but marked as inactive.

- Task actions that are associated with tasks from DB2 Task Center:

Table 9. Restrictions for task actions from DB2 Task Center

Task action	Restrictions
Run task	Only the first Run task task action associated with the task will be imported.
Enable schedule of	Not supported.
Disable schedule of	Not supported.
Delete this task	Not supported.

- The success code sets that are used by the DB2 Task Center when running tasks are ignored by the job manager.
- If the tools catalog database contains a task that was previously imported to the Data Studio web console and you choose to import the task again, the task is saved as a new job with a new job ID.
- Contact lists are not imported from the DB2 Task Center.

Procedure

To import tasks from the DB2 Task Center:

1. Open the Data Studio web console in a web browser.
2. To open the Import Tasks page, from the **Open** menu, click **Product Setup** > **Import Tasks**.
3. Follow the instructions on the Import Tasks page to start importing tasks. You must specify a valid tools catalog database that contains the DB2 Task Center metadata, and then select the tasks to import. Only supported tasks from the tools catalog database are enabled in the Import Tasks page.

Results

If the task is imported successfully, a new job is created for the imported task in the job manager with a job name that is identical to the task name of the imported task. The job name is prefixed by "TC_*toolsdb*_" where *toolsdb* is the name of the DB2 tools database. The script of the imported task is not modified.

If the imported task is associated with a schedule in the Task Center, a new schedule is created for the corresponding job by the job manager and the tools catalog database is associated with the schedule by default. The schedule date format for the imported task is converted to the job manager schedule format.

What to do next

If the job that was generated from the imported task is not associated with a schedule, create a schedule and add the job to the schedule.

Diagramming access plans with Visual Explain

You can generate a diagram of the current access plan for an SQL or XPATH statement to find out how your data server processes the statement. You can use the information available from the graph to tune your SQL statements for better performance.

Before you begin

If you want to create access plan diagrams for DB2 for z/OS, you must configure the DB2 subsystem that you are using. The steps are identical to the steps for configuring a subsystem for use with the no-charge tuning features that are in IBM Data Studio.

Restriction: For IBM Informix® Dynamic Server, Visual Explain cannot explain SELECT statements that contain parameter markers or host variables.

About this task

You can use Visual Explain to:

- View the statistics that were used at the time of optimization. You can then compare these statistics to the current catalog statistics to help you determine whether rebinding the package might improve performance.
- Determine whether or not an index was used to access a table. If an index was not used, Visual Explain can help you determine which columns might benefit from being indexed.
- Obtain information about each operation in the access plan, including the total estimated cost and number of rows retrieved (cardinality).

Procedure

To generate the diagram of the current access plan for a query:

1. Optional: Set preferences for how Visual Explain operates and for how it displays diagrams.
2. Follow one of these steps:
   - In the Data Project Explorer, right-click an SQL statement, SQL stored procedure, or SQL user-defined function, and select **Open Visual Explain**.
   - In the Data Source Explorer, right-click a view or right-click an SQL stored procedure or SQL user-defined function that contains an INSERT, UPDATE, DELETE, or SELECT statement. Select **Open Visual Explain**. If the workbench finds more than one SQL statement or XQUERY statement, the workbench uses the first statement.
   - In an SQL, Routine, or Java™ editor, highlight and right-click the INSERT, UPDATE, DELETE, or SELECT statement, XPATH, or XQUERY statement and select **Open Visual Explain**.

     Attempts to open Visual Explain from an SQL statement in a Java editor fail if the SQL statement contains variables that are declared in your application. For example, this SQL statement cannot be analyzed by Visual Explain because of the two variables in the predicate:

     ```
     select count(*), sum(order.price)
     from order
     where order.date > var_date_1
     and order.date < var_date_2
     ```

     However, after you bind or deploy the application, you can use InfoSphere® Optim Query Tuner or the single-query tuning features in Data Studio to capture the SQL statement from a DB2 package or from the dynamic statement cache and then tune it.

   Note: Visual Explain is disabled or throws an exception if the selected SQL statement or object is not explainable. Only the SQL statements in the following list can be explained by Visual Explain:

   - For DB2 for Linux, UNIX, and Windows: CALL, Compound SQL (Dynamic), DELETE, INSERT, MERGE, REFRESH, SELECT, SELECT INTO, SET INTEGRITY, UPDATE, VALUES, or VALUES INTO.
   - For DB2 for z/OS: SELECT, INSERT, or the searched form of an UPDATE or DELETE statement.
3. On the first page of the wizard, specify the terminator of the SQL, XPATH, or XQUERY statement that you want to diagram the access plan for.
4. Optional: On the first page of the wizard, you can also specify settings for various options.

   a. Specify whether you want to store the collected explain data in explain tables. If you choose this option, Visual Explain does not have to collect explain data the next time that you want to diagram the access plan for the same statement.
   b. Specify the directory that you want Visual Explain to use as a working directory.
   c. If IBM Support needs a trace, specify whether to trace the creation of the diagram of the access plan and whether to trace the collection of the explain data.
   d. Specify whether to save your settings as the defaults for all diagrams that you create with Visual Explain. You can change these defaults with the Preferences window.
5. On the second page of the wizard, set values for the special registers to customize the runtime environment to influence the collection of explain data.

   When Visual Explain runs the statement to gather explain data, it uses the values that you specify.

   Attention: Please be aware of the following information regarding DB2 data servers.

   - **For DB2 for z/OS:** If you specify different values for CURRENT SCHEMA and CURRENT SQLID, Visual Explain searches for explain tables that are qualified by the value of CURRENT SQLID. If Visual Explain does not find explain tables that are qualified by the value of CURRENT SQLID, Visual Explain attempts to create the explain tables under that value.
   - **For DB2 for Linux, UNIX, and Windows:** If you change the value of CURRENT SCHEMA to a value that contains special characters, you must delimit the value with single quotation marks.
   - **For DB2 for Linux, UNIX, and Windows:** Select the **Collect column and column group statistics** check box if you want Visual Explain to collect detailed statistics about clustered columns and columns that participate in a GROUP BY clause.
6. Optional: On the second page of the wizard, specify whether to save your settings as the defaults for all diagrams that you create with Visual Explain. You can change these defaults with the Preferences window.
7. Click **Finish** to close the wizard and to generate the diagram.

Results

The workbench displays the diagram in the Access Plan Diagram view. In this view, you can navigate through the diagram, view descriptions of the nodes in the diagram, and search for nodes.

Diagrams of access plans

When DB2 processes a query, the DB2 optimizer generates several alternative plans for accessing the requested data. The optimizer estimates the execution cost of each plan and chooses the lowest-cost plan to execute. This plan is called the access plan.

Visual Explain graphically displays the access plan for any explainable statement. This display is called an access plan diagram, and it illustrates how DB2 accesses the data for a specified SQL statement.

The access plan diagram consists of nodes and lines that connect those nodes. The nodes represent data sources, operators, SQL statements, and query blocks. Nodes can have only one parent node, but they can have unlimited child nodes. The arrows on the edges indicate the direction of the flow. Usually, a table node is at the bottom of the graph, and the access plan proceeds upward from there.

Some operations in the access plan, such as nested loop joins or index scans, are represented in the graph by groups of nodes, which are called constructs. Many of these constructs have a defining node that indicates the operation. For example, the HBJOIN node indicates that a hybrid join operation is taking place, but the entire hybrid join is represented in the graph by a group of nodes. This group of nodes represents all of the other data sources and operations that are involved in the hybrid join.

Query blocks

An SQL statement can consist of several subqueries, which are represented in the access plan diagram by query blocks.

The subquery can be a SELECT, INSERT, UPDATE, or DELETE. A subquery can contain other subqueries in the FROM clause, the WHERE clause, or a subselect of a UNION or UNION ALL. A subquery within another subquery is called a child subquery. A subquery that contains another subquery is called a parent subquery. This parent-child relationship can be represented by a tree hierarchy.

If a subquery references at least one column of its parent subquery or of any parent subqueries that are higher up in the tree hierarchy, the subquery is a correlated subquery; otherwise it is a non-correlated subquery. A non-correlated subquery can run at the same time as the highest parent subquery that is also non-correlated. This highest parent subquery is called the "do-at-open parent subquery" in terms of its relationship to the non-correlated subquery. The execution of a correlated subquery is bound to the execution of its parent subquery. Such relationships between the relative executions of parents and children can be represented by separate trees hierarchies in the access plan graph.

Non-correlated subquery
For a non-correlated subquery, the query block node is connected to the right of the query block node for the highest parent subquery that is also non-correlated.

Correlated subquery
For a correlated subquery, the query block node is connected to the part within its parent subquery where the correlated subquery is executed.

Setting preferences for Visual Explain

Use the Preferences window to set default values for settings that determine how Visual Explain operates and how it displays diagrams.

Procedure

To set preferences for Visual Explain:

1. Select **Window > Preferences**.
2. In the tree view of the Preferences window, select **Data > Visual Explain**.
3. On the Visual Explain page, set the following options:
   a. Specify whether to launch the Visual Explain wizard when you right-click an SQL statement, view, stored procedure, or user-defined function and select **Visual Explain**. The wizard allows you to override preferences. If you clear this option, Visual Explain uses the preferences.

   b. If your project is associated with a DB2 data server, specify whether Visual Explain saves in the explain tables the explain data that it collects for the statement.

4. On the **Query Explain Settings** page, specify default values for special registers. Changing these values modifies how Visual Explain gathers explain data to use when generating the access plan diagram.

   Attention: Please be aware of the following information regarding DB2 data servers.

   - **For DB2 for z/OS:** If you specify different values for CURRENT SCHEMA and CURRENT SQLID, Visual Explain searches for explain tables that are qualified by the value of CURRENT SQLID. If Visual Explain does not find explain tables that are qualified by the value of CURRENT SQLID, Visual Explain attempts to create the explain tables under that value.
   - **For DB2 for Linux, UNIX, and Windows:** If you change the value of CURRENT SCHEMA to a value that contains special characters, you must delimit the value with single quotation marks.
   - **For DB2 for Linux, UNIX, and Windows:** Select the **Collect column and column group statistics** check box if you want Visual Explain to collect detailed statistics about clustered columns and columns that participate in a GROUP BY clause.
5. On the **Viewer** page, change various behaviors and colors of diagrams.
6. On the **Nodes** page, change the default appearance of nodes. You can change the text, color, and shape of the different types of nodes. You can also choose whether to highlight selected nodes, shadow nodes, or show information about nodes when you move your mouse cursor over them.

Part 2. Client-to-server communications

Configuring client-to-server communications for the IBM data server client and DB2 database server products requires and understanding of the components and type of connections.

Components and scenarios

The basic components involved in client-to-server communications are described in the following section:

- **Client.** This refers to the initiator of the communications. This role can be filled by any of the following DB2 products or components:
  - IBM Data Server Driver Package
  - IBM Data Server Client or IBM Data Server Runtime Client.
  - DB2 Connect Personal Edition: This product is a superset of the IBM Data Server Client.
  - a DB2 server product: A DB2 server is a superset of the Data Server Client.
- **Server.** This refers to the receiver of the communications request from the client. This role is normally filled by a DB2 for Linux, UNIX, and Windows server product. When DB2 Connect products are present, the term *server* can also mean a DB2 server on a midrange or mainframe platform.
- **Communications protocol.** This refers to the protocol used to send data between the client and server. The DB2 product supports several protocols:
  - TCP/IP. A further distinction can be made between the version: TCP/IPv4 or TCP/IPv6.
  - Named Pipes. This option is available on Windows only.
  - IPC (interprocess communications). This protocol is used for local connections.

There are also some additional components encountered in some environments:

- **DB2 Connect gateway.** This refers to a DB2 Connect server product that provides a gateway by which IBM data server client can connect to DB2 servers on midrange and mainframe products.
- **LDAP (Lightweight Directory Access Protocol).** In an LDAP-enabled environment, it is not necessary to configure client-to-server communications. When a client attempts to connect to a database, if the database does not exist in the database directory on the local machine then the LDAP directory is searched for information required to connect to the database.

The following scenarios illustrate examples of situations covered by client-to-server communications:

- Data Server Client establishes communications with a DB2 server using TCP/IP.
- Data Server Runtime Client establishes communications with a DB2 server using Named Pipes on a Windows network.
- DB2 server establishes communications with another DB2 server via some communications protocol.
- Data Server Client establishes communications with a mainframe DB2 server via a DB2 Connect server using TCP/IP.

When setting up a server to work with development environments (such as IBM Data Studio), you might encounter error message SQL30081N at the initial DB2 connection. A possible root cause is that the firewall at the remote database server has prevented the connection from being established. In this case, verify the firewall is properly configured to accept connection requests from the client.

Types of connections

Generally speaking, references to setting up client-to-server communications refer to *remote connections*, rather than *local connections*.

A *local connection* is a connection between a database manager instance and a database managed by that instance. In other words, the CONNECT statement is issued from the database manager instance to itself. Local connections are distinctive because no communications setup is required and IPC (interprocess communications) is used.

A *remote connection* is one where the client issuing the CONNECT statement to a database is in a different location from the database server. Commonly, the client and server are on different machines. However, remote connections are possible within the same machine if the client and server are in different instances.

Another less common type of connection is a *loopback connection*. This is a type of remote connection where the connection is configured from a DB2 instance (the client) to the same DB2 instance (the server).

Configuration of client-to-server communications

You can configure client-to-server communications by using the command line tools which consist of the Command Line Processor (CLP), the **db2cfexp** (configuration export) command, and the **db2cfimp** (configuration import) command.

Use the following table to identify the appropriate configuration method.

Table 10. Tools and methods for configuring a client-to-server connection

Type of configuration task	CLP
Configure a client by entering information manually	Configure client-to-server connections by using the **CATALOG TCPIP/TCPIP4/TCPIP6 NODE command** and the **CATALOG DATABASE** command.
Use the connection settings for one client as the basis for configuring additional clients	1. Create a client profile by issuing the **db2cfexp** command. 2. Configure database connections using a client profile by issuing the **db2cfimp** command.

Note: Use *Profiles* to configure client-to-server communications. The types of profiles are:

- A *client profile* is a file that contains settings for a client. Settings can include:
  - Database connection information (including CLI or ODBC settings).
  - Client settings (including database manager configuration parameters and DB2 registry variables).
  - CLI or ODBC common parameters.

- A *server profile* is similar to a client profile but contains settings for a server.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for configuring automatic maintenance. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Chapter 5. Supported combinations of clients, drivers and server levels

Various versions of a client or driver can connect to different versions of a server. This includes support for earlier versions and support for accessing DB2 databases on midrange and mainframe servers.

DB2 client levels required for IBM DB2 pureScale Feature

For your application to make full use of DB2 pureScale features, your DB2 client must be at certain release levels:

Server version	Client version	Features available
Version 9.8 or later	Version 9.7, Fix Pack 1 or later	Transaction-level and connection-level workload balancing Automatic client reroute based on workload Client affinities
Version 9.8 or later	Version 9.1, Version 9.5, or Version 9.7 (before Fix Pack 1)	Connection-level workload balancing (transaction-level workload balancing is not available) Automatic client reroute based on workload

Combinations of DB2 Version 9.1, DB2 Version 9.5, DB2 Version 9.7, and DB2 Version 10.1 clients and servers

Generally, DB2 Version 9.1, DB2 Version 9.5, and DB2 Version 9.7 clients can access a remote DB2 Version 10.1 server. However, if different versions of a client and a DB2 server are located on the same system, local client-to-server connections using Interprocess Communication (IPC) are not supported. Instead, you can establish a connection as a remote connection (called a *loopback connection*) by using TCP/IP.

The following clients and drivers can access a DB2 Version 9.7, DB2 Version 9.5 or DB2 Version 9.1 server:

- IBM Data Server Client Version 10.1
- IBM Data Server Runtime Client Version 10.1
- IBM Data Server Driver Package Version 10.1
- IBM Data Server Driver for ODBC and CLI Version 10.1

However, when a later-level client accesses an earlier-level server, the functionality of the later level of the client is not available to the server. For example, IBM Data Server Driver Package Version 10.1 can access a DB2 Version 9.5 server; however, DB2 Version 10.1 functionality is not available to the server.

Note: DB2 Version 9.1 reached end of support on April 30, 2012. For more support lifecycle information, see http://www-01.ibm.com/software/data/support/lifecycle/. For continued Version 9.1 support, a service extension is required.

Combinations of DB2 Version 10.1 and DB2 products on midrange and mainframe platforms

DB2 Version 10.1 servers support access from the following clients on midrange and mainframe platforms:

- DB2 for z/OS and OS/390® Version 8 or later
- DB2 for i5/OS™ Version 5 or later
- DB2 for VM and VSE Version 7

The following clients and drivers can access a DB2 Connect Version 9.7, Version 9.5 or Version 9.1 server:

- IBM Data Server Client Version 10.1
- IBM Data Server Runtime Client Version 10.1
- IBM Data Server Driver Package Version 10.1
- IBM Data Server Driver for ODBC and CLI Version 10.1

Note: DB2 Connect Version 9.1 reached end of support on April 30, 2012. For more support lifecycle information, see http://www-01.ibm.com/software/data/support/lifecycle/. For continued Version 9.1 support, a service extension is required.

Chapter 6. Communication protocols supported

This topic identifies the supported protocols for connecting from an IBM data server client to a DB2 server. This includes:

- connecting from IBM data server client to midrange or mainframe hosts using DB2 Connect products.
- connecting from mid range or mainframe platforms to databases on DB2 for Linux, UNIX, and Windows.

The TCP/IP protocol is supported on all platforms on which DB2 for Linux, UNIX, and Windows is available. Both TCP/IPv4 and TCP/IPv6 are supported. IPv4 addresses have a four-part structure, for example, `9.11.22.314`. IPv6 addresses have an eight-part name, where each part consists of 4 hex digits delimited by a colon. Two colons (::) represents one or more sets of zeros. For example, `2001:0db8:4545:2::09ff:fef7:62dc`.

DB2 database products support the SSL protocol and accept SSL requests from applications that use the IBM Data Server Driver for JDBC and SQLJ (type 4 connectivity), IBM Data Server Driver for ODBC and CLI and IBM Data Server Driver Package. Refer to Configuring Secure Sockets Layer (SSL) support in a DB2 instance.

In addition, the Windows Named Pipes protocol is supported on Windows networks. To administer a DB2 database remotely, you must connect using TCP/IP.

Chapter 7. Supported LDAP client and server configurations

The following table summarizes the supported LDAP client and server configurations.

IBM Tivoli Directory Server is an LDAP Version 6.2 server and is available for Windows, AIX, Solaris, Linux, and HP-UX and is shipped as part of the base operating system on AIX and System i®, and with OS/390 Security Server.

The DB2 database supports IBM LDAP client on AIX, Solaris, HP-UX 11.11, Windows, and Linux.

Microsoft Active Directory server is an LDAP Version 3 server and is available as part of the Windows 2000 Server and Windows Server 2003 family of operating systems.

The Microsoft LDAP Client is included with the Windows operating system.

Table 11. Supported LDAP client and server configurations

Supported LDAP Client and Server Configurations	**IBM Tivoli Directory server**	**Microsoft Active Directory server**	**Sun One LDAP server**
IBM LDAP Client	Supported	Supported	Supported
Microsoft LDAP/ADSI Client	Supported	Supported	Supported

Note: When running on Windows operating systems, the DB2 database manager supports using either the IBM LDAP client or the Microsoft LDAP client. To explicitly select the IBM LDAP client, use the **db2set** command to set the DB2LDAP_CLIENT_PROVIDER registry variable to "IBM". The Microsoft LDAP Client is included with the Windows operating system.

Chapter 8. Discovery of administration servers, instances, and databases

To configure connections to a remote computer, you can use an existing directory service such as Lightweight Directory Access Protocol (LDAP).

Known Discovery allows you to discover instances and databases on systems that are known to your client, and add new systems so that their instances and databases can be discovered. Search Discovery provides all of the facilities of Known Discovery and adds the option to allow your local network to be searched for other DB2 database servers.

To have a system support Known Discovery, set the **discover** parameter in the DAS configuration file to `KNOWN`. To have the system support both Known and Search Discovery, set the **discover** parameter in the DAS configuration file to `SEARCH` (this is the default). To prevent discovery of a system, and all of its instances and databases, set this parameter to `DISABLE`. Setting the **discover** parameter to `DISABLE` in the DAS configuration file, prevents discovery of the system.

Note: The TCP/IP host name returned to a client by Search Discovery is the same host name that is returned by your DB2 server system when you enter the **hostname** command. On the client, the IP address that this host name maps to is determined by either the TCP/IP domain name server (DNS) configured on your client computer or, if no DNS is configured, a mapping entry in the client's `hosts` file. If you have multiple adapter cards configured on your DB2 server system, you must ensure that TCP/IP is configured on the server to return the correct hostname, and that the DNS or local client's `hosts` file, maps the hostname to the IP address desired.

On the client, enabling Discovery is also done using the **discover** parameter; however, in this case, the **discover** parameter is set in the client instance (or server acting as a client) as follows:

- `KNOWN`

  KNOWN discovery is used to retrieve instance and database information associated with systems that are already known to your local system. New systems can be added using the **Add Systems** functionality provided in the tools. When the **discover** parameter is set to `KNOWN`, you will not be able to search the network.
- `SEARCH`

  Enables all of the facilities of Known Discovery, and enables local network searching. This means that any searching is limited to the local network.

  The **Other Systems (Search the network)** icon only appears if this choice is made. This is the default setting.
- `DISABLE`

  Disables Discovery. In this case, the **Search the network** option is not available in the Add Database Wizard.

Note: The **discover** parameter defaults to `SEARCH` on all client and server instances. The **discover** parameter defaults to `SEARCH` on all DB2 administration servers (DAS).

Discovering and hiding server instances and databases

You might have multiple instances, and multiple databases within these instances, on a server system. You might want to hide some of these from the Discovery process.

Procedure

- To allow clients to discover server instances on a system, set the **discover_inst** database manager configuration parameter in each server instance on the system to ENABLE (this is the default value).

  Set this parameter to DISABLE to hide this instance and its databases from Discovery.
- To allow a database to be discovered from a client, set the **discover_db** database configuration parameter to ENABLE (this is the default value).

  Set this parameter to DISABLE to hide the database from Discovery.

  Note: If you want an instance to be discovered, **discover** must also be set to KNOWN or SEARCH in the DAS configuration file.

  Note: If you want a database to be discovered, the **discover_inst** parameter must also be enabled in the server instance.

Chapter 9. Configuring DB2 server communications (TCP/IP)

This task describes how to configure TCP/IP communications on your DB2 server using the DB2 Command Line Processor (CLP). Communication protocols on the DB2 server must be configured in order for your DB2 server to accept inbound requests from remote DB2 clients.

Before you begin

Before you configure TCP/IP communications for an instance on your DB2 server:

- Ensure that the TCP/IP protocol is functional on the DB2 server. TCP/IP must also be functional on the DB2 client to establish a connection.
- Identify either a Connection Service name *and* Connection Port, or just a Connection Port.

  Connection Service Name and Connection Port

  The service name is used to update the Service name (**svcename**) parameter in the database manager configuration file at the server. When a Connection Service name is specified, the services file must be updated with the same Service name, a port number, and the protocol. The Service name is arbitrary but must be unique within the services file. A sample value for the service name could be `server1`. If you are using DB2 Enterprise Server Edition in a partitioned format, ensure that the port number does not conflict with the port numbers used by the Fast Communications Manager (FCM).

  The Connection port must be unique within the `services` file. A sample value for the port number and protocol could be `3700/tcp`.

  Connection Port

  The Service name (**svcename**) parameter in the database manager configuration file at the server can be updated with a port number. If this is the case, it is not necessary to update the `services` file. If you are using DB2 Enterprise Server Edition in a partitioned format, ensure that the port number does not conflict with the port numbers used by the Fast Communications Manager (FCM) or any other applications on the system. A sample value for the port number could be `3700`.

About this task

Most protocols are automatically detected and configured when you set up DB2 database systems using the DB2 Setup wizard. Perform the current task if:

- You deselected the TCP/IP communication protocol when you set up the DB2 database system using the DB2 Setup wizard.
- You added the TCP/IP communication protocol to your network after you set up the DB2 database system using the DB2 Setup wizard.
- The TCP/IP communication protocol was not detected by the DB2 Setup wizard.
- You installed a DB2 database product using the **db2_install** command or the payload file method.

Procedure

To configure TCP/IP communications for a DB2 instance:

1. Update the `services` file on the server. Refer to "Updating the services file on the server for TCP/IP communications."
2. Update the database manager configuration file on the server. Refer to "Updating the database manager configuration file on the server for TCP/IP communications."
3. Set communication protocols for a DB2 instance. Refer to "Setting communication protocols for a DB2 instance" on page 89.

Updating the services file on the server for TCP/IP communications

This task is part of the main task of *Configuring TCP/IP communications for a DB2 instance*.

About this task

The TCP/IP services file specifies the ports that server applications can listen on for client requests. If you specified a service name in the **`svcename`** field of the DBM configuration file, the services file must be updated with the service name to port number/protocol mapping. If you specified a port number in the **`svcename`** field of the DBM configuration file, the services file does *not* need to be updated.

Update the `services` file and specify the ports that you want the server to listen on for incoming client requests. The default location of the `services` file depends on the operating system:

Linux and UNIX operating systems
`/etc/services`

Windows operating systems
`%SystemRoot%\system32\drivers\etc\services`

Procedure

Using a text editor, add the Connection entry to the services file. For example:

```
db2c_db2inst1  3700/tcp # DB2 connection service port
```

where:

db2c_db2inst1
represents the connection service name

3700 represents the connection port number

tcp represents the communication protocol that you are using

Updating the database manager configuration file on the server for TCP/IP communications

This task is part of the main task of configuring TCP/IP communications for a DB2 instance.

About this task

You must update the database manager configuration file with the service name (**`svcename`**) parameter.

Procedure

To update the database manager configuration file:

1. Log on to the system as a user with System Administrative (SYSADM) authority.
2. If you are using a UNIX operating system, set up the instance environment:
   ```
   . INSTHOME/sqllib/db2profile    (for Bash, Bourne or Korn shell)
   source INSTHOME/sqllib/db2cshrc (for C shell)
   ```
3. Start the DB2 command line processor (CLP).
4. Update the database manager configuration file with the Service name (**svcename**) parameter by entering the following commands:
   ```
      update database manager configuration using svcename
        [service_name | port_number]
      db2stop
      db2start
   ```

   where:
   - *service_name* is the service name reserved in the `services` file
   - *port_number* is the corresponding port number for the *service_name*, or a free port number if the *service_name* is not reserved

   If a service name is being specified, the **svcename** used must match the Connection Service name specified in the `services` file.

   After the database manager is stopped and started again, view the database manager configuration file to ensure that these changes have taken effect. View the database manager configuration file by entering the following command:
   ```
      get database manager configuration
   ```

Setting communication protocols for a DB2 instance

Setting communication protocols for a DB2 instance is part of the main task of configuring TCP/IP or SSL communications for a DB2 instance.

Before you begin

To perform this task you require SYSADM authority.

About this task

The **DB2COMM** registry variable allows you to set communication protocols for the current DB2 instance. If the **DB2COMM** registry variable is undefined or set to null, no protocol connection managers are started when the database manager is started.

The **DB2COMM** registry variable can be set with the following keywords:

tcpip starts TCP/IP support

ssl starts SSL support

Procedure

To set the communication protocol for the instance:

Enter the **db2set DB2COMM** command from the DB2 command window:

```
db2set DB2COMM=tcpip
```

Example

For example, to set the database manager to start connection managers for the TCP/IP communication protocols, enter the following command:

```
db2set DB2COMM=tcpip
db2stop
db2start
```

Chapter 10. Configuring client-to-server connections

This task describes how to configure a connection from an IBM data server client to a remote database server using the command line processor (CLP).

Before you begin

Before you configure a client to server connection, ensure:

- Network communications is set up between the machine with the IBM data server client and the machine with the DB2 server. One way to verify this for the TCP/IP protocol is to use the **ping** command.
- The DB2 server is configured to work on the network. This is normally done as part of installing and configuring the DB2 server product.

Procedure

Separate topics are provided to guide you through each of the following steps. Some steps have a version for each supported protocol:

1. Identify the communication parameter values for the remote database server.
2. If you are using TCP/IP, you have the option to update the client's hosts file and services file with communication parameter values for the remote database server. For more details, see "Updating hosts and services files for TCP/IP connections" on page 92. This step does not apply to Named Pipes.
3. Catalog the server node from the client. Instructions are provided for each communications protocol:

   "Cataloging a TCP/IP node from a client using the CLP" on page 93

   "Cataloging a Named Pipes node from a client using the CLP"
4. Catalog the database that you want to connect to on the client. For more details, see "Cataloging a database" on page 94.
5. Test the client-to-server connection. For more details, see "Testing the client-to-server connection using the CLP" on page 96.

Cataloging a Named Pipes node from a client using the CLP

Cataloging a Named Pipes node adds an entry to the client's node directory to describe the remote node. This entry specifies the chosen alias (*node_name*), the remote server's workstation name (*computer_name*), and the instance (*instance_name*) that the client will use to access the remote DB2 server.

Procedure

To catalog a Named Pipes node on an IBM data server client, type the following command in the command line processor (CLP):

```
db2 => catalog npipe node node_name
db2 => remote computer_name instance instance_name

db2 => terminate
```

Example

To catalog a remote node called db2node that is located on a server called server1 in the db2 instance, use:

```
db2 => db2 catalog npipe node db2node remote server1 instance db2

db2 => terminate
```

Updating hosts and services files for TCP/IP connections

This task explains when and how to update the hosts file and services file on the client with communication parameter values for the remote database server. This task is optional for connections using TCP/IP and does not apply to connections using Named Pipes. This task is part of the larger task of configuring client-to-server connection using the CLP.

About this task

You need to update the hosts file if you want to establish a connection to the remote database server using its hostname and your network does not contain a DNS (domain name server) that can be used to resolve that hostname to an IP address. This step is not required if you want to refer to the remote database server using its IP address.

You need to update the services file if you want to specify a *connection service* name when establishing a connection to the remote database server. A *connection service* is an arbitrary name that represents the connection port number. This step is not required if you want to refer to the remote database server's port number.

Procedure

- To update the hosts file on the client to resolve the remote server's hostname to its IP address:
  1. Use a text editor to add an entry to the hosts file for the server's IP address. For example:

```
9.26.13.107                           myserver   # IPv4 address for myserver
2002:91a:519:13:210:83ff:feff:ca71     myserver   # IPv6 address for myserver
```

where:

9.26.13.107
represents the *IPv4 ip_address*

2002:91a:519:13:210:83ff:feff:ca71
represents the *IPv6 ip_address*

myserver
represents the *hostname*

represents a comment describing the entry

Note: Note that IPv6 entries are not needed if your host does not belong on an IPv6 network. For hosts in mixed IPv4 and IPv6 networks, an alternate method is to assign different host names for IPv4 and IPv6 addresses. For example:

```
9.26.13.107                        myserver     # IPv4 address for myserver
9.26.13.107                        myserveripv4 # IPv4 address for myserver
2002:91a:519:13:210:83ff:feff:ca71 myserveripv6 # IPv6 address for myserver
```

If the server is not in the same domain as the IBM data server client, you must provide a fully qualified domain name such as `myserver.spifnet.ibm.com`, where `spifnet.ibm.com` represents the domain name.

- To update the `services` file on the client to resolve a service name to the remote server's port number:
  1. Using a text editor, add the Connection Service name and port number to the `services` file. For example:

     ```
     server1  50000/tcp  # DB2 connection service port
     ```

     where:

     server1
     represents the Connection Service name

     50000
     represents the connection port number (50000 is the default)

     tcp
     represents the communication protocol that you are using

     # represents the beginning of a comment that describes the entry

Example

The following table lists the location of the `hosts` file and `services` file referred to in the preceding procedures.

Table 12. Location of the hosts file and services file

Operating System	Directory
Windows 2000 XP/Windows Server 2003	*%SystemRoot%*`\system32\drivers\etc` where *%SystemRoot%* is an environment variable defined on the system.
Linux or UNIX	`/etc`

Cataloging a TCP/IP node from a client using the CLP

Cataloging a TCP/IP node adds an entry to the Data Server Client node directory that describes the remote node. This entry specifies the chosen alias (*node_name*), the *hostname* (or *ip_address*), and the *svcename* (or *port_number*) that the client uses to access the remote host.

Before you begin

You must have System Administrative (SYSADM) or System Controller (SYSCTRL) authority, or have the **catalog_noauth** option set to `ON`. You cannot catalog a node using root authority.

Procedure

To catalog a TCP/IP node:

1. Log on to the system as a user with System Administrative (SYSADM) or System Controller (SYSCTRL) authority.
2. If you are using a Linux or UNIX client, set up the instance environment. Run the startup script:

   For bash, Bourne or Korn shell

```
. INSTHOME/sqllib/db2profile
```

For C shell

```
source INSTHOME/sqllib/db2cshrc
```

where *INSTHOME* represents the home directory of the instance.

3. Start the DB2 command line processor. On Windows, issue the **db2cmd** command from a command prompt. On Linux or UNIX, issue the **db2** command from a command prompt.
4. Catalog the node by entering the following commands in the command line processor:

```
db2 => catalog tcpip node node_name remote hostname|ip_address
  server service_name|port_number [remote_instance instance_name]
  [system system_name] [ostype os_type]

db2 => terminate
```

where:

- *node_name* represents a local nickname you can set for the computer that has the database you want to catalog.
- *remote_instance* represents the name of the server instance on which the database resides.
- *system_name* represents the DB2 system name that is used to identify the server.
- *ostype_name* represents the operating system type of the server.

Note:

a. The **terminate** command is needed to refresh the directory cache.
b. Although **remote_instance**, **system**, and **ostype** are optional, they are required for users who want to use the DB2 tools.
c. The *service_name* used on the client does not have to be the same as the one on the server. However, the port numbers that they map to *must* match
d. While not shown here, the **catalog tcpip node** command provides the option to explicitly specify the version of IP, namely IPv4 or IPv6.

Example

To catalog a node that you want to call db2node on a remote server myserver.ibm.com that is using port number 50000, you would enter the following from a **db2** prompt:

```
db2 => catalog tcpip node db2node remote myserver server 50000
DB20000I  The CATALOG TCPIP NODE command completed successfully.
DB21056W  Directory changes may not be effective until the directory cache is
refreshed.

db2 => terminate
DB20000I  The TERMINATE command completed successfully.
```

Cataloging a database

This task describes how to catalog a database from a client by using the command line processor (CLP).

Before you begin

Before a client application can access a remote database, the database must be cataloged on the client. When you create a database, the database is automatically cataloged on the server with a database alias that is the same as the database name, unless a different database alias was specified.

The information in the database directory, along with the information in the node directory (unless you are cataloging a local database where a node is not needed), is used on the IBM data server client to establish a connection to the remote database.

- You require a valid DB2 user ID. DB2 does not support using root authority to catalog a database.
- You must have System Administrative (SYSADM) or System Controller (SYSCTRL) authority, or have the **`catalog_noauth`** option set to `ON`.
- You need the following information when cataloging a *remote* database:
  - Database name
  - Database alias
  - Node name
  - Authentication type (optional)
  - Comment (optional)

  Refer to the parameter values worksheet for cataloging a database for more information about these parameters and to record the values that you use.
- The following parameter values are applicable when cataloging a *local* database:
  - Database name
  - Drive
  - Database alias
  - Authentication type (optional)
  - Comment (optional)

  Local databases can be uncataloged and recataloged at any time.

Procedure

To catalog a database on the client:

1. Log on to the system with a valid DB2 user ID.
2. If you are using the DB2 database on a Linux or UNIX platform, set up the instance environment. Run the startup script:

   For bash, Bourne or Korn shell

   ```
   . INSTHOME/sqllib/db2profile
   ```

   For C shell

   ```
   source INSTHOME/sqllib/db2cshrc
   ```

   where: *INSTHOME* represents the home directory of the instance.
3. Start the DB2 command line processor. On Windows operating systems, issue the **`db2cmd`** command from a command prompt. On Linux or UNIX, issue the **`db2`** command from a command prompt.
4. Catalog the database by entering the following commands in the command line processor:

```
db2 => catalog database database_name as database_alias at
   node node_name [ authentication auth_value ]
```

where:

- *database_name* represents the name of the database you want to catalog.
- *database_alias* represents a local nickname for the database you want to catalog.
- *node_name* represents a nickname you can set for the computer that has the database you want to catalog.
- *auth_value* specifies the type of authentication that takes place when connecting to the database. This parameter defaults to the authentication type specified on the server. Specifying an authentication type can result in a performance benefit. Examples of valid values include: `SERVER`, `CLIENT`, `SERVER_ENCRYPT`, `KERBEROS`, `DATA_ENCRYPT`, `GSSPLUGIN` and `SERVER_ENCRYPT_AES`.

Example

To catalog a remote database called SAMPLE so that it has the local database alias MYSAMPLE, on the node DB2NODE using authentication `SERVER`, enter the following commands:

```
db2 => catalog database sample as mysample at node db2node
       authentication server
db2 => terminate
```

Testing the client-to-server connection using the CLP

Before you begin

After cataloging the node and the database, connect to the database to test the connection. Before testing the connection:

- The database node and database must be cataloged.
- The values for *userid* and *password* must be valid for the system on which they are authenticated. The authentication parameter on the client is be set to match the value on the server or it can be left unspecified. If an authentication parameter is not specified, the client will default to SERVER_ENCRYPT. If the server does not accept SERVER_ENCRYPT, then the client retries using the value returned from the server. If the client specifies an authentication parameter value that doesn't match what is configured on the server, you will receive an error.
- The database manager must be started with the correct protocol defined in the **DB2COMM** registry variable. If it is not started, then you can start the database manager by entering the **db2start** command on the database server.

Procedure

To test the client to server connection:

1. If you are using a Linux or UNIX platform, set up the instance environment. Run the startup script:

   For bash, Bourne or Korn shell

   ```
   . INSTHOME/sqllib/db2profile
   ```

   For C shell

   ```
   source INSTHOME/sqllib/db2cshrc
   ```

   where: *INSTHOME* represents the home directory of the instance.

2. Start the DB2 command line processor. On Windows, issue the **db2cmd** command from a command prompt. On Linux or UNIX, issue the **db2** command from a command prompt.
3. Type the following command on the client to connect to the remote database:

   ```
   db2 => connect to database_alias user userid
   ```

   For example, enter the following command:

   ```
   connect to mysample user jtris
   ```

   You will be prompted to enter your password.

Example

If the connection is successful, you receive a message showing the name of the database to which you have connected. A message similar to the following is given:

```
Database Connection Information
Database server = DB2 9.1.0
SQL authorization ID = JTRIS
Local database alias = mysample
```

You can now work with the database. For example, to retrieve a list of all the table names listed in the system catalog table, enter the following SQL statement:

```
select tabname from syscat.tables
```

What to do next

When you are finished using the database connection, enter the **connect reset** command to end the database connection.

Exporting and importing a profile

If you did not use a configuration profile when you installed your DB2 product using the response file that was created by the response file generator, you can create a configuration file and import it to another workstation.

Procedure

1. To create a configuration profile, enter the **db2cfexp** command specifying the fully qualified name of the target export file. The resulting profile contains only configuration information associated with the current DB2 database instance.
2. To import the configuration profile, you can:
   - Use the **db2cfimp** command
   - Use a response file by uncommenting the keyword DB2.CLIENT_IMPORT_PROFILE and specify the *filename* as the export file

Chapter 11. Configuring LDAP connections

In an LDAP-enabled environment, the directory information about DB2 servers and databases is stored in the LDAP directory. When a new database is created, the database is automatically registered in the LDAP directory.

During a database connection, the client accesses the LDAP directory to retrieve the required database and protocol information and uses this information to connect to the database.

Use the DB2 CLP commands in the LDAP environment to:

- Manually catalog a database in the LDAP directory.
- Register a database cataloged in LDAP as an ODBC data source.
- Configure CLI/ODBC information about the LDAP server.
- Remove a database cataloged in the LDAP directory.

Cataloging an LDAP node

A node name for the DB2 server must be specified when registering the server in LDAP. Applications use the node name to attach to the database server.

Procedure

- If you require a different node name, such as when the node name is hard-coded in an application, use the **CATALOG LDAP NODE** command to make the change. For example:

  ```
  db2 catalog ldap node ldap_node_name
     as new_alias_name
  ```
- To uncatalog a LDAP node, use the **UNCATALOG LDAP NODE** command. For example:

  ```
  db2 uncatalog ldap node ldap_node_name
  ```

Registering DB2 servers

Each DB2 server instance must be registered in LDAP to publish the protocol configuration information that is used by the client applications to connect to the DB2 server instance.

About this task

When registering an instance of the database server, you must specify a *node name*. The node name is used by client applications when they connect or attach to the server. You can catalog another alias name for the LDAP node by using the **CATALOG LDAP NODE** command.

Note: If you are working in a Windows domain environment, then during installation the DB2 server instance is automatically registered in the Active Directory with the following information:

```
nodename: TCP/IP hostname
protocol type: TCP/IP
```

If the TCP/IP hostname is longer than eight characters, it is truncated to eight characters.

The **REGISTER** command appears as follows:

```
db2 register db2 server in ldap
   as ldap_node_name
   protocol tcpip
```

The protocol clause specifies the communication protocol to use when connecting to this database server.

When creating an instance for DB2 Enterprise Server Edition that includes multiple physical machines, the **REGISTER** command must be invoked once for each computer. Use the **rah** command to issue the **REGISTER** command on all computers.

Note: The same *ldap_node_name* cannot be used for each computer since the name must be unique in LDAP. You will want to substitute the hostname of each computer for the *ldap_node_name* in the **REGISTER** command. For example:

```
rah ">DB2 REGISTER DB2 SERVER IN LDAP AS <> PROTOCOL TCPIP"
```

The "<>" is substituted by the hostname on each computer where the **rah** command is run. In the rare occurrence where there are multiple DB2 Enterprise Server Edition instances, the combination of the instance and host index can be used as the node name in the **rah** command.

The **REGISTER** command can be issued for a remote DB2 server. To do so, you must specify the remote computer name, instance name, and the protocol configuration parameters when registering a remote server. The command can be used as follows:

```
db2 register db2 server in ldap
   as ldap_node_name
   protocol tcpip
   hostname host_name
   svcename tcpip_service_name
   remote remote_computer_name
   instance instance_name
```

The following convention is used for the computer name:

- If TCP/IP is configured, the computer name must be the same as the TCP/IP hostname.

When running in a high availability or failover environment, and using TCP/IP as the communication protocol, the cluster IP address must be used. Using the cluster IP address allows the client to connect to the server on either computer without having to catalog a separate TCP/IP node for each computer. The cluster IP address is specified using the hostname clause, shown as follows:

```
db2 register db2 server in ldap
   as ldap_node_name
   protocol tcpip
   hostname n.nn.nn.nn
```

where *n.nn.nn.nn* is the cluster IP address.

To register the DB2 server in LDAP from a client application, call the db2LdapRegister API.

Registering databases

During the creation of a database within an instance, the database is automatically registered in LDAP. Registration allows remote client connection to the database without having to catalog the database and node on the client computer.

When a client attempts to connect to a database, if the database does not exist in the database directory on the local computer then the LDAP directory is searched.

About this task

If the name exists in the LDAP directory, the database is still created on the local computer but a warning message is returned stating the naming conflict in the LDAP directory. For this reason, you can manually catalog a database in the LDAP directory. The user can register databases on a remote server in LDAP by using the **CATALOG LDAP DATABASE** command. When registering a remote database, you specify the name of the LDAP node that represents the remote database server. You must register the remote database server in LDAP using the **REGISTER DB2 SERVER IN LDAP** command before registering the database.

Procedure

- To register a database manually in LDAP, use the **CATALOG LDAP DATABASE** command:

```
db2 catalog ldap database dbname
   at node node_name
   with "My LDAP database"
```

- To register a database in LDAP from a client application, call the db2LdapCatalogDatabase API.

Creating LDAP users

When using the IBM Tivoli directory, you must define an LDAP user before you can store user-level information in LDAP. You can create an LDAP user by creating an LDIF file to contain all attributes for the user object, then run the LDIF import utility to import the object into the LDAP directory.

About this task

The DB2 database system supports setting DB2 registry variables and CLI configuration at the user level. (This is not available on the Linux and UNIX platforms.) User level support provides user-specific settings in a multi-user environment. An example is Windows Terminal Server where each logged on user can customize his or her own environment without interfering with the system environment or another user's environment.

The **LDIF** utility for the IBM Tivoli Directory Server is **LDIF2DB**.

LDIF file containing the attributes for a person object appears similar to the following:

```
   File name: newuser.ldif

 dn: cn=Mary Burnnet, ou=DB2 Development, ou=Toronto, o=ibm, c=ca
 objectclass: ePerson
 cn: Mary Burnnet
 sn: Burnnet
 uid: mburnnet
```

```
userPassword: password
telephonenumber: 1-416-123-4567
facsimiletelephonenumber: 1-416-123-4568
title: Software Developer
```

Following is an example of the **LDIF** command to import an LDIF file using the IBM **LDIF** import utility:

```
LDIF2DB -i newuser.ldif
```

Note:

1. You must run the **LDIF2DB** command from the LDAP server.
2. You must grant the required access (ACL) to the LDAP user object so that the LDAP user can add, delete, read, and write to his own object. To grant ACL for the user object, use the LDAP Directory Server Web Administration tool.

Configuring LDAP users for DB2 applications

When you use the Microsoft LDAP client, the LDAP user is the same as the operating system user account. However, when you use the IBM LDAP client, before you use the DB2 database manager, you must configure the LDAP user distinguished name (DN) and password for the current logged on user.

Procedure

To configure the LDAP user distinguished name (DN) and password, use the **db2ldcfg** utility:

```
db2ldcfg -u userDN -w password --> set the user's DN and password
         -r                    --> clear the user's DN and password
```

For example:

```
db2ldcfg -u "cn=Mary Burnnet,ou=DB2 Development,ou=Toronto,o=ibm,c=ca"
         -w password
```

Setting DB2 registry variables at the user level in the LDAP environment

Under the LDAP environment, the DB2 profile registry variables can be set at the user level which allows a user to customize their own DB2 environment.

About this task

DB2 for Linux, UNIX, and Windows has a caching mechanism. The DB2 profile registry variables at the user level are cached on the local computer.

The cache is refreshed when:

- You update or reset a DB2 registry variable at the user level.
- You issue the command to refresh the LDAP profile variables at the user level:

  ```
  db2set -ur
  ```

Procedure

To set the DB2 profile registry variables at the user level, use the **-ul** option:

```
db2set -ul variable=value
```

Note: This is not supported on AIX or Solaris operating systems.
If the **-ul** parameter is specified, DB2 for Linux, UNIX, and Windows always reads from the cache for the DB2 registry variables.

Deregistering DB2 servers

Deregistration of an instance from LDAP also removes all the node, or alias, objects, and the database objects referring to the instance.

About this task

Deregistration of the DB2 server on either a local or a remote computer requires the LDAP node name be specified for the server.

Procedure

To deregister the DB2 server from LDAP:

- From the command line, use the **DEREGISTER** command:

  ```
  db2 deregister db2 server in ldap
     node node_name
  ```

- From a client application, call the db2LdapDeregister API.

Results

When the DB2 server is deregistered, any LDAP node entry and LDAP database entries referring to the same instance of the DB2 server are also uncataloged.

Deregistering the database from the LDAP directory

The database is automatically deregistered from LDAP when the database is dropped, or the owning instance is deregistered from LDAP.

Procedure

To deregister a database from the LDAP directory:

- From the command line, use the **UNCATALOG LDAP DATABASE** command:

  ```
  db2 uncatalog ldap database dbname
  ```

- From a client application, call the **db2LdapUncatalogDatabase** API.

Chapter 12. Configuring IBM Data Server Drivers

Use the db2dsdriver.cfg configuration file to configure the IBM Data Server Driver. This configuration file contains database directory information and configuration keywords to set up connections to supported databases through ODBC, CLI, .NET, OLE DB, or open source (PHP or Ruby) applications. For a complete list of configuration keywords, see "IBM Data Server Driver configuration keywords" at the following URL: http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.swg.im.dbclient.config.doc/doc/c0054698.html.

The IBM data server driver configuration file is an XML file that is based on the `db2dsdriver.xsd` schema definition file. You can specify a customized path for the IBM data server driver configuration file by using one of the following methods:

- Use the **DB2DSDRIVER_CFG_SOURCE_PATH** response file keyword to specify the path during the IBM data server product installation.
- Set the **DB2DSDRIVER_CFG_PATH** registry variable to the file path.

You can associate the keywords globally, meaning with all database connections, or you can associate the keywords with a specific database source name (DSN) or database connection. You can also use the configuration file to enable a high availability connection to supported databases.

You can use the IBM data server driver configuration file with embedded SQL applications, ODBC, CLI, .NET, OLE DB, PHP, or Ruby drivers. However, the IBM data server driver configuration file is not required to use embedded SQL applications, ODBC, CLI, .NET, OLE DB, PHP, or Ruby drivers. The applications can function without the IBM data server driver configuration file. However, instead of specifying the database name, host, port, and configuration parameters in your applications, you can use the configuration file with defined aliases.

The IBM data server driver configuration file supports a set of XML tags that are in lowercase and do not include underscores (_). XML tag attributes, which are where you specify IBM data server driver configuration keywords, can contain uppercase, lowercase, and underscore (_) characters.

The sample IBM data server driver configuration file, named `db2dsdriver.cfg.sample`, is included with IBM data server products.

Copying existing database directory information into the db2dsdriver configuration file

You can populate the `db2dsdriver.cfg` configuration file with existing database directory information.

Before you begin

You must have an existing Version 9.7 IBM Data Server Client or IBM Data Server Runtime Client installed.

About this task

The `db2dsdriver.cfg` configuration file configures the behavior of DB2 CLI, ODBC, open source, or .NET applications by using keywords. The keywords are associated with the database alias name, and affect all the applications that access the database.

If you have an IBM Data Server Client or IBM Data Server Runtime Client, you can copy the existing database directory information into the `db2dsdriver.cfg` configuration file by using the **`db2dsdcfgfill`** command. Using this command, the configuration file is populated based on the contents of the local database directory, node directory, and Database Connection Services (DCS) directory of a specific database manager instance.

Restrictions

None.

Procedure

To copy existing database directory information from an IBM Data Server Client or IBM Data Server Runtime Client into the `db2dsdriver.cfg` configuration file:

Enter the **`db2dsdcfgfill`** command. For example, `db2dsdcfgfill -i` *`instance_name`* `-o` *`output_path`*. The parameter **`-o`** *output-path* indicates the path where the `db2dsdriver.cfg` configuration file is created. For information about the location of the `db2dsdriver.cfg` file, see the topic about that file.

Part 3. Physical design and business rules implementation

Physical database design consists of defining database objects and implementing business rules.

You can create the following database objects in a DB2 database:

- Tables
- Constraints
- Indexes
- Triggers
- Sequences
- Views
- Usage lists

You ca use Data Definition Language (DDL) statements or tools such as IBM Data Studio to create these database objects. The DDL statements are generally prefixed by the keywords CREATE or ALTER.

Understanding the features and functionality that each of these database objects provides is important to implement a good database design that meets your current business's data storage needs while remaining flexible enough to accommodate expansion and growth over time.

Chapter 13. Databases

A DB2 database is a *relational database*. The *database* stores all data in tables that are related to one another. Relationships are established between tables such that data is shared and duplication is minimized.

A *relational database* is a database that is treated as a set of tables and manipulated in accordance with the relational model of data. It contains a set of objects used to store, manage, and access data. Examples of such objects are tables, views, indexes, functions, triggers, and packages. Objects can be either defined by the system (built-in objects) or defined by the user (user-defined objects).

A *distributed relational database* consists of a set of tables and other objects that are spread across different but interconnected computer systems. Each computer system has a relational database manager to manage the tables in its environment. The database managers communicate and cooperate with each other in a way that allows a given database manager to execute SQL statements on another computer system.

A *partitioned relational database* is a relational database whose data is managed across multiple database partitions. This separation of data across database partitions is transparent to most SQL statements. However, some data definition language (DDL) statements take database partition information into consideration (for example, **`CREATE DATABASE PARTITION GROUP`**). DDL is the subset of SQL statements used to describe data relationships in a database.

A *federated database* is a relational database whose data is stored in multiple data sources (such as separate relational databases). The data appears as if it were all in a single large database and can be accessed through traditional SQL queries. Changes to the data can be explicitly directed to the appropriate data source.

Designing databases

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

The first step is to describe the system that you want to represent. For example, if you were creating a database for publishing system, the system would contain several types of entities, such as books, authors, editors, and publishers. For each of these entities, there are certain pieces of information, or attributes, that you must record:

- *Books:* titles, ISBN, date published, location, publisher, ...
- *Authors:* name, address, phone and fax numbers, email address, ...
- *Editors:* name, address, phone and fax numbers, email address, ...
- *Publishers:* name, address, phone and fax numbers, email address, ...

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

There are three types of relationships between the entities in a database:

One-to-one relationships
In this type of relationship, each instance of an entity relates to only one instance of another entity. Currently, no one-to-one relationships exist in the scenario described previously.

One-to-many relationships
In this type of relationship, each instance of an entity relates to one or more instances of another entity. For example, an author could have written multiple books, but certain books have only one author. This is the most common type of relationship modeled in relational databases.

Many-to-many relationships
In this type of relationship, many instances of a given entity relate to one or more instances of another entity. For example, co-authors could write a number of books.

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

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

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

You can:

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

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

Creating databases

You create a database using the **CREATE DATABASE** command. To create a database from a client application, call the sqlecrea API. All databases are created with the default storage group IBMSTOGROUP, unless you specify otherwise. Automatic storage managed table spaces use storage groups for their storage definitions.

Before you begin

The DB2 database manager must be running. Use the **db2start** command to start the database manager.

It is important to plan your database, keeping in mind the contents, layout, potential growth, and how it will be used before you create it. After it has been created and populated with data, changes can be made.

The following database privileges are automatically granted to PUBLIC: CREATETAB, BINDADD, CONNECT, IMPLICIT_SCHEMA, and SELECT on the system catalog views. However, if the **RESTRICTIVE** option is present, no privileges are automatically granted to PUBLIC. For more information about the **RESTRICTIVE** option, see the **CREATE DATABASE** command.

Restrictions

- Storage paths cannot be specified using relative path names; you must use absolute path names. The storage path can be up to 175 characters long.
- On Windows operating systems, the database path must be a drive letter only, unless the **DB2_CREATE_DB_ON_PATH** registry variable is set to YES.
- If you do not specify a database path using the **DBPATH ON** clause of the **CREATE DATABASE** command, the database manager uses the first storage path specified for the **ON** clause for the database path. (On Windows operating systems, if this clause is specified as a path, and if the **DB2_CREATE_DB_ON_PATH** registry variable is not set to YES, you receive a SQL1052N error message.) If no **ON** clause is specified, the database is created on the default database path that is specified in the database manager configuration file (**dftdbpath** parameter). The path is also used as the location for the single storage path associated with the database.
- For partitioned databases, you must use the same set of storage paths on each database partition (unless you use database partition expressions).
- Database partition expressions are not valid in database paths, whether you specify them explicitly by using the **DBPATH ON** clause of the **CREATE DATABASE** command, or implicitly by using a database partition expression in the first storage path.
- A storage group must have at least one storage path associated with it.

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

About this task

When you create a database, each of the following tasks are done for you:

- Setting up of all the system catalog tables that are needed by the database
- Allocation of the database recovery log
- Creation of the database configuration file and the default values are set
- Binding of the database utilities to the database

Procedure

- To create a database from a client application, call the sqlecrea API.
- To create a database using the command line processor, issue the **CREATE DATABASE** command.

For example, the following command creates a database called PERSON1, in the default location, with the associated comment "Personnel DB for BSchiefer Co".

```
CREATE DATABASE person1
  WITH "Personnel DB for BSchiefer Co"
```

- To create a database using IBM Data Studio, right-click the instance on which you want to create the database and select the task assistant to the create it. For more information, see IBM Data Studio: Administering databases with task assistants.

Example

Example 1: Creating a database on a UNIX or Linux operating system:

To create a database named TESTDB1 on path /DPATH1 using /DATA1 and /DATA2 as the storage paths defined to the default storage group IBMSTOGROUP, use the following command:

```
CREATE DATABASE TESTDB1 ON '/DATA1','/DATA2' DBPATH ON '/DPATH1'
```

Example 2: Creating a database on a Windows operating system, specifying both storage and database paths:

To create a database named TESTDB2 on drive D:, with storage on E:\DATA, use the following command:

```
CREATE DATABASE TESTDB2 ON 'E:\DATA' DBPATH ON 'D:'
```

In this example, E:\DATA is used as both the storage path defined to the default storage group IBMSTOGROUP and the database path.

Example 3: Creating a database on a Windows operating system, specifying only a storage path:

To create a database named TESTDB3 with storage on drive F:, use the following command:

```
CREATE DATABASE TESTDB3 ON 'F:'
```

In this example, F: is used as both the storage path defined to the default storage group IBMSTOGROUP and the database path.

If you specify a directory name such as F:\DATA for the storage path, the command fails, because:

1. When **DBPATH** is not specified, the storage path -- in this case, F:\DATA -- is used as the database path
2. On Windows, the database path can only be a drive letter (unless you change the default for the **DB2_CREATE_DB_ON_PATH** registry variable from NO to YES).

If you want to specify a directory as the storage path on Windows operating systems, you must also include the **DBPATH ON** drive clause, as shown in Example 2.

Example 4: Creating a database on a UNIX or Linux operating system without specifying a database path:

To create a database named TESTDB4 with storage on /DATA1 and /DATA2, use the following command:

```
CREATE DATABASE TESTDB4 ON '/DATA1','/DATA2'
```

In this example, /DATA1 and /DATA2 are used as the storage paths defined to the default storage group IBMSTOGROUP and /DATA1 is the database path.

What to do next

Configuration Advisor

The Configuration Advisor helps you to tune performance and to balance memory requirements for a single database per instance by suggesting which configuration parameters to modify and providing suggested values for them. The Configuration Advisor is automatically invoked by default when you create a database.

You can override this default so that the configuration advisor is not automatically invoked by using one of the following methods:

- Issue the **CREATE DATABASE** command with the **AUTOCONFIGURE APPLY NONE** parameter.
- Set the **DB2_ENABLE_AUTOCONFIG_DEFAULT** registry variable to NO:

  ```
  db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
  ```

  However, if you specify the **AUTOCONFIGURE** parameter with the **CREATE DATABASE** command, the setting of this registry variable is ignored.

Also, the following automatic features are enabled by default when you create a database:

- Automatic storage
- Automatic background statistics collection
- Automatic real-time statistics collection
- Self-tuning memory (single-partition environments)

Event Monitor

At the same time a database is created, a detailed deadlocks event monitor is also created. As with any monitor, there is extra processing time and resources associated with this event monitor. If you do not want the detailed deadlocks event monitor, then the event monitor can be dropped by using the command:

```
 DROP EVENT MONITOR db2detaildeadlock
```

To limit the amount of disk space that this event monitor consumes, the event monitor deactivates, and a message is written to the administration notification log, once it has reached its maximum number of output files. Removing output files that are no longer needed allows the event monitor to activate again on the next database activation.

Remote databases

You can create a database in a different, possibly remote, instance. To create a database at another (remote) database partition server, you must first attach to that server. A database connection is temporarily established by the following command during processing:

```
CREATE DATABASE database_name AT DBPARTITIONNUM options
```

In this type of environment, you can perform instance-level administration against an instance other than your default instance, including remote instances. For instructions on how to do this, see the **db2iupdt** (update instance) command.

Database code pages

By default, databases are created in the UTF-8 (Unicode) code set.

To override the default code page for the database, it is necessary to specify the required code set and territory when creating the database. See the **CREATE DATABASE** command or the sqlecrea API for information about setting the code set and territory.

Converting a nonautomatic storage database to use automatic storage

You can convert an existing nonautomatic storage database to use automatic storage by using the CREATE STOGROUP statement to define the default storage group within a database.

Before you begin

You must have a storage location that you can identify with a path (for Windows operating systems, a path or a drive letter) available to use as a storage path for your automatic storage table spaces.

Restrictions

- Once you have created a storage group, you cannot drop all storage groups for a database.
- Only DMS table spaces can be converted to use automatic storage.

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

About this task

Databases that are created specifying the AUTOMATIC STORAGE NO clause of the **CREATE DATABASE** command do not have storage groups associated with them. Instead, storage is associated with the table spaces for the database. When you define a storage group for a database, existing table spaces are not automatically converted to use automatic storage. By default, only future table spaces that you create are automatic storage table spaces. You must use the ALTER TABLESPACE statement to convert existing table spaces to use automatic storage.

Procedure

You can convert an existing database to an automatic storage database by using the CREATE STOGROUP statement to create a storage group within it.

To create a storage group within a database, use the following statement:

```
CREATE STOGROUP sg ON storagePath
```

where *sg* is the storage group and *storagePath* is the path you want to use for automatic storage table spaces.

Example

Example 1: Converting a database on UNIX or Linux operating systems

Assume that the database EMPLOYEE is a nonautomatic storage database, and that `/data1/as` and `/data2/as` are the paths you want to use for automatic storage table spaces. To convert EMPLOYEE to an automatic storage database, create a storage group with `/data1/as` and `/data2/as` as paths:

```
CREATE STOGROUP sg ON '/data1/as', '/data2/as'
```

Example 2: Converting a database on Windows operating systems

Assume that the database SALES is a nonautomatic storage database, and that `F:\DB2DATA` and `G:` are the paths you want to use for automatic storage table spaces. To convert SALES to an automatic storage database, create a storage group with `F:\DB2DATA` and `G:` as paths:

```
CREATE STOGROUP sg ON 'F:\DB2DATA', 'G:'
```

What to do next

If you have existing DMS table spaces that you want to convert to use automatic storage, use the ALTER TABLESPACE statement with the MANAGED BY AUTOMATIC STORAGE clause to change them. If you do not specify the USING STOGROUP clause, then the table space uses the storage paths in the designated default storage group.

Once you have created a storage group you can create automatic storage table spaces in which to store tables, indexes and other database objects by using the CREATE TABLESPACE statement.

Chapter 14. Buffer pools

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

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

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

How buffer pools are used

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

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

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

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

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

Designing buffer pools

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

Before you create a new buffer pool, resolve the following items:
- What buffer pool name do you want to use?
- Whether the buffer pool is to be created immediately or following the next time that the database is deactivated and reactivated?
- Whether the buffer pool should exist for all database partitions, or for a subset of the database partitions?
- What page size you want for the buffer pool? See "Buffer pool page sizes".
- Whether the buffer pool will be a fixed size, or whether the database manager will automatically adjust the size of the buffer pool in response to your workload? It is suggested that you allow the database manager to tune your buffer pool automatically by leaving the SIZE parameter unspecified during buffer pool creation. For details, see the SIZE parameter of the "CREATE BUFFERPOOL statement" and "Buffer pool memory considerations" on page 119.
- Whether you want to reserve a portion of the buffer pool for block based I/O? For details, see: "Block-based buffer pools for improved sequential prefetching".

Relationship between table spaces and buffer pools

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

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

Buffer pool page sizes

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

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

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

Buffer pool memory considerations

Memory requirements

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

Buffer pool memory protection

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

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

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

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

Buffer pool hit ratios

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

Buffer pool memory protection (AIX running on POWER6)

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

Storage keys is a new feature in IBM Power6 processors and the AIX operating system that allows the protection of ranges of memory using hardware keys at a kernel thread level. Storage key protection reduces buffer pool memory corruption problems and limits errors that might halt the database. Attempts to illegally access the buffer pool by programming means cause an error condition that the database manager can detect and deal with.

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

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

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

DB2_MEMORY_PROTECT registry variable

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

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

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

Creating buffer pools

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

Before you begin

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

About this task

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

Procedure

To create a buffer pool using the command line:

1. Get the list of buffer pool names that exist in the database. Issue the following SQL statement:
   ```
   SELECT BPNAME FROM SYSCAT.BUFFERPOOLS
   ```
2. Choose a buffer pool name that is not currently found in the result list.
3. Determine the characteristics of the buffer pool you are going to create.
4. Ensure that you have the correct authorization ID to run the CREATE BUFFERPOOL statement.
5. Issue the CREATE BUFFERPOOL statement. A basic CREATE BUFFERPOOL statement is:
   ```
       CREATE BUFFERPOOL buffer-pool-name
         PAGESIZE 4096
   ```

Results

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

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

Example

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

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

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

For more information, see the "CREATE BUFFERPOOL statement".

Modifying buffer pools

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

Before you begin

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

About this task

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

- Enable self tuning for a buffer pool, allowing the database manager to adjust the size of the buffer pool in response to your workload.
- Modify the block area of the buffer pool for block-based I/O.
- Add this buffer pool definition to a new database partition group.
- Modify the size of the buffer pool on some or all database partitions.

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

1. To get the list of the buffer pool names that already exist in the database, issue the following statement:
   ```
   SELECT BPNAME FROM SYSCAT.BUFFERPOOLS
   ```
2. Choose the buffer pool name from the result list.
3. Determine what changes must be made.
4. Ensure that you have the correct authorization ID to run the ALTER BUFFERPOOL statement.

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

   With DEFERRED, the changes to the buffer pool will not be applied until the database is reactivated. Reserved memory space is not needed; the database manager allocates the required memory from the system at activation time.
5. Use the ALTER BUFFERPOOL statement to alter a single attribute of the buffer pool object. For example:
   ```
   ALTER BUFFERPOOL buffer pool name SIZE number of pages
   ```

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

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

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

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

Dropping buffer pools

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

You cannot drop the IBMDEFAULTBP buffer pool.

About this task

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

Procedure

To drop buffer pools, use the DROP BUFFERPOOL statement.

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

Chapter 15. Table spaces

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

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

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

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

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

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

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

Figure 5 on page 126 illustrates the relationship between tables and table spaces within a database, and the containers associated with that database.

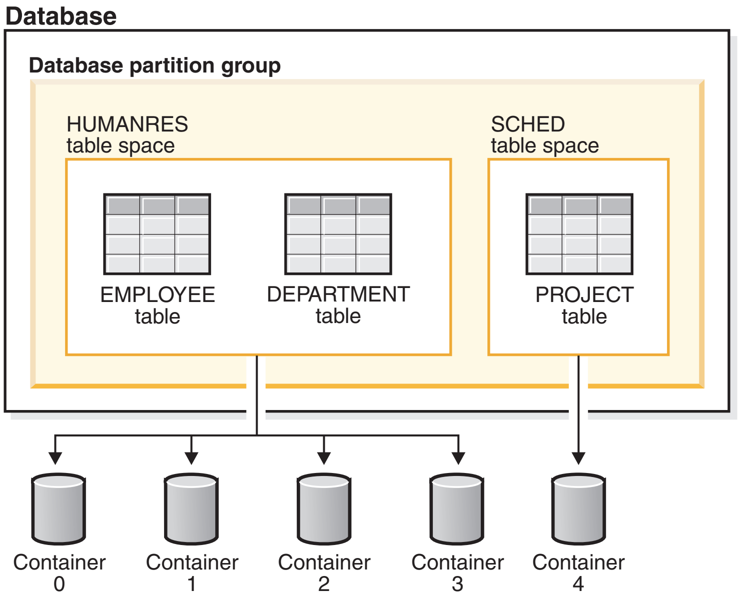


Figure 5. Table spaces and tables in a database

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

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

Figure 6 on page 127 shows the HUMANRES table space with an extent size of two 4 KB pages, and four containers, each with a small number of allocated extents. The DEPARTMENT and EMPLOYEE tables both have seven pages, and span all four containers.

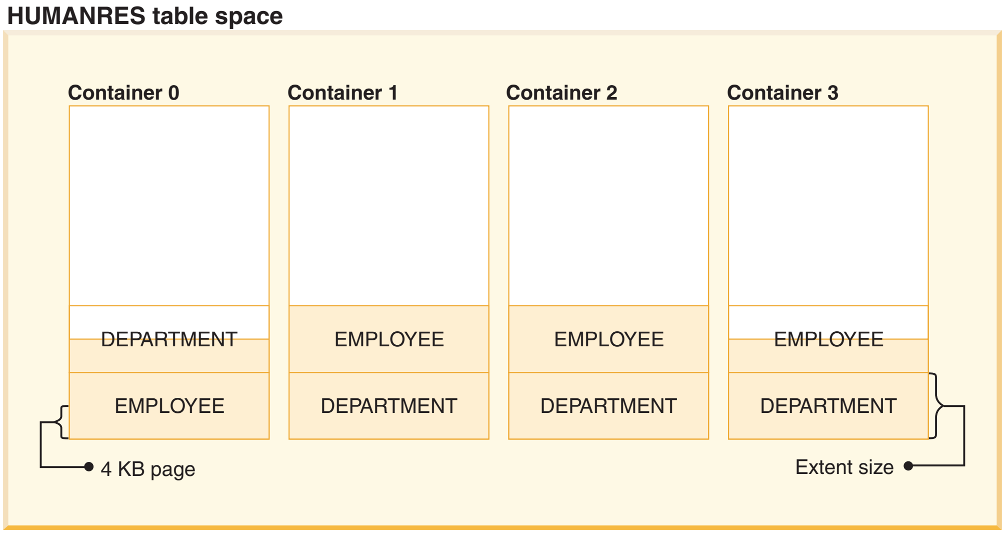


Figure 6. Containers and extents in a table space

Table spaces for system, user and temporary data

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

A database must contain at least three table spaces:
- A *catalog table space*
- One or more *user table spaces*
- One or more *temporary table spaces.*

Catalog table spaces

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

User table spaces

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

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

The page size of a table space determines the maximum row length or number of columns that you can have in a table. The documentation for the CREATE TABLE statement shows the relationship between page size, and the maximum row size and column count. Before Version 9.1, the default page size was 4 KB. In Version 9.1 and following, the default page size can be one of the other supported values. The default page size is declared when creating a new database. Once the default page size has been declared, you are still free to create a table space with one page size for the table, and a different table space with a different page size for long or LOB data. If the number of columns or the row size exceeds the limits for a table space's page size, an error is returned (SQLSTATE 42997).

Temporary table spaces

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

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

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

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

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

Types of table spaces

You can set up table spaces in different ways depending on how you choose to manage their storage.

The three types of table spaces are known as:

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

Each can be used together in any combination within a database

Automatic storage table spaces

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

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

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

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

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

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

How automatic storage table spaces manage storage expansion

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

When an automatic storage table space is created, the database manager creates a container on each of the storage paths of the storage group it is defined to use (where space permits). Once all of the space in a table space is consumed, the database manager automatically grows the size of the table space by extending existing containers or by adding a new stripe set of containers.

Storage for automatic table spaces is managed at the storage group level; that is, you add storage to the database's *storage groups*, rather than to table spaces as you do with DMS table spaces. When you add storage to a storage group used by the table space, the automatic storage feature will create new containers as needed to accommodate data. However, table spaces that already exist will not start consuming storage on the new paths immediately. When a table space needs to grow, the database manager will first attempt to extend those containers in the last *range* of the table space. A range is all the containers across a given stripe set. If this is successful, applications will start using that new space. However, if the attempt to extend the containers fails, as might happen when one or more of the file systems are full, for example, the database manager will attempt to create a new stripe set of containers. Only at this point does the database manager consider using the newly added storage paths for the table space. Figure 7 illustrates this process.

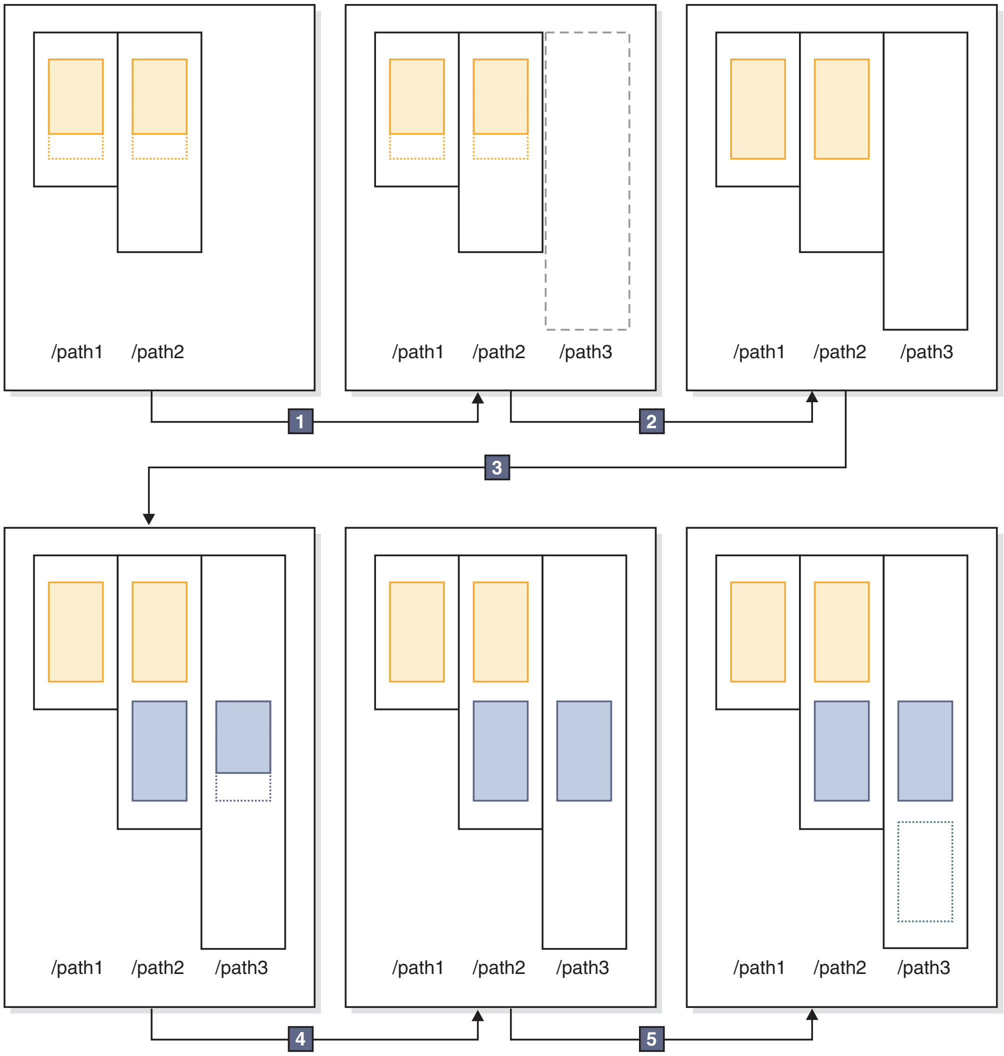


Figure 7. How automatic storage adds containers as needed

In the preceding diagram:

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

If you want to have the automatic storage table space start using the newly added storage path immediately, you can perform a rebalance, using the REBALANCE clause of the ALTER TABLESPACE command. If you rebalance your table space, the data will be reallocated across the containers and stripe sets in the newly-added storage. This is illustrated in Figure 8.

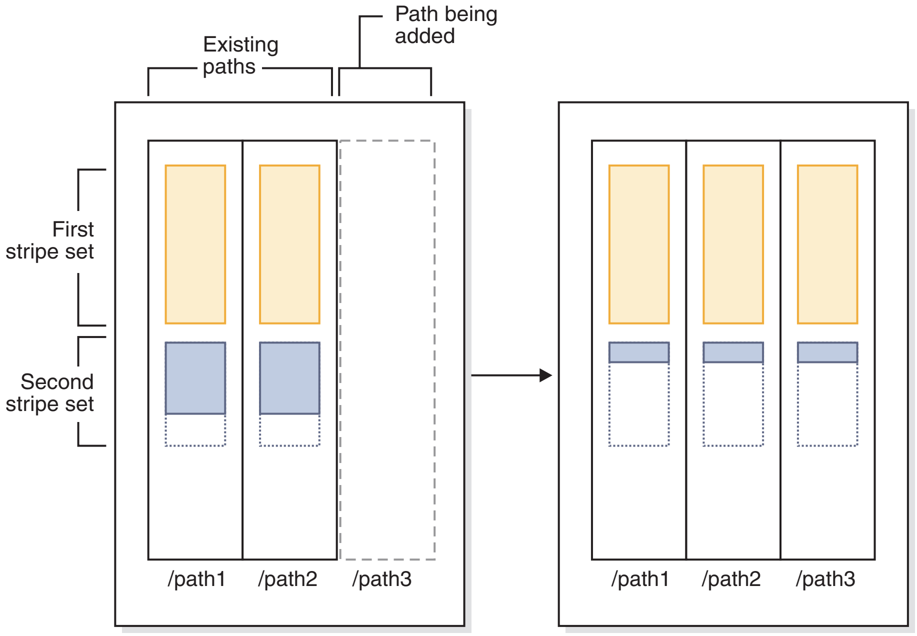


Figure 8. Results of adding new storage and rebalancing the table space

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

Container names in automatic storage table spaces

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

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

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

where:

storage path
: Is a storage path associated with a storage group

instance name
: Is the instance under which the database was created

database name
: Is the name of the database

NODE####
: Is the database partition number (for example, NODE0000)

T#######
: Is the table space ID (for example, T0000003)

C#######
: Is the container ID (for example, C0000012)

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

  CAT System catalog table space

  TMP System temporary table space

  UTM User temporary table space

  USR User or regular table space

  LRG Large table space

Example

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

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

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

```
LIST TABLESPACE CONTAINERS FOR 10 SHOW DETAIL

            Tablespace Containers for Tablespace 10

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

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

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

Converting table spaces to use automatic storage

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

Before you begin

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

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

Procedure

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

- **Alter a single table space.** This method keeps the table space online but involves a rebalance operation that takes time to move data from the non-automatic storage containers to the new automatic storage containers.
  1. Specify the table space that you want to convert to automatic storage. Indicate which storage group you want the table space to use. Issue the following statement:
     ```
     ALTER TABLESPACE tbspc1 MANAGED BY AUTOMATIC STORAGE USING STOGROUP sg_medium
     ```
     where `tbspc1` is the table space and `sg_medium` is the storage group it is defined in.
  2. Move the user-defined data from the old containers to the storage paths in the storage group `sg_medium` by issuing the following statement:
     ```
     ALTER TABLESPACE tbspc1 REBALANCE
     ```
     Note: If you do not specify the REBALANCE option now and issue the ALTER TABLESPACE statement later with the REDUCE option, your automatic storage containers will be removed. To recover from this problem, issue the ALTER TABLESPACE statement, specifying the REBALANCE option.
  3. To monitor the progress of the rebalance operation, use the following statement:
     ```
     SELECT * from table (MON_GET_REBALANCE_STATUS( 'tbspc1', -2))
     ```
- **Use a redirected restore operation.** When the redirected restore operation is in progress, you cannot access the table spaces being converted. For a full database redirected restore, all table spaces are inaccessible until the recovery is completed.
  1. Run the **RESTORE DATABASE** command, specifying the **REDIRECT** parameter. If you want to convert a single table space, also specify the **TABLESPACE** parameter:
     ```
     RESTORE DATABASE database_name TABLESPACE (table_space_name) REDIRECT
     ```
  2. Run the **SET TABLESPACE CONTAINERS** command, specifying the **USING AUTOMATIC STORAGE** parameter, for each table space that you want to convert:
     ```
     SET TABLESPACE CONTAINERS FOR tablespace_id USING AUTOMATIC STORAGE
     ```

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

   ```
   RESTORE DATABASE database_name CONTINUE
   ```

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

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

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

Example

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

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

   ```
   RESTORE DATABASE testdb REDIRECT
   ```

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

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

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

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

   ```
   RESTORE DATABASE testdb CONTINUE
   ```

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

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

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

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

The table space high water mark

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

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

Tip: Extents are indexed from 0. So the high water mark is the *last page of the highest allocated extent* + 1.
Practically speaking, it's virtually impossible to determine the high water mark yourself; there are administrative views and table functions that you can use to

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

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

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

Example

Figure 9 shows a series of allocated extents in a table space.

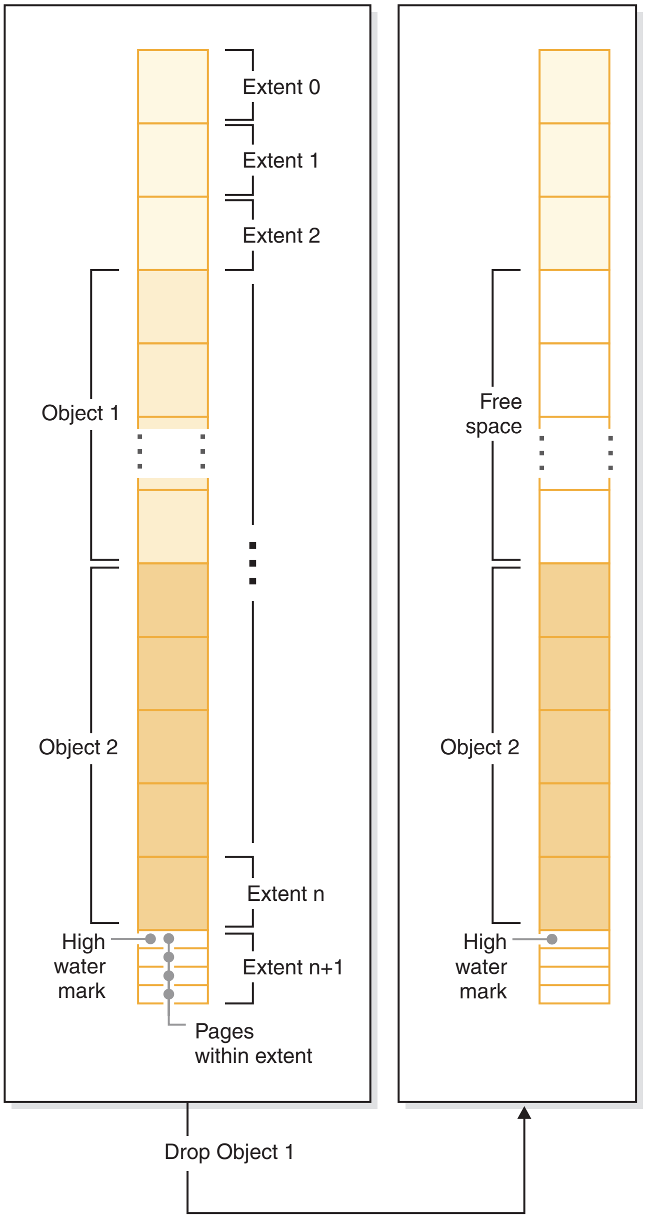


Figure 9. High water mark

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

Reclaimable storage

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

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

During the extent consolidation process, extents that contain data are moved to unused extents below the high water mark. After extents are moved, if free extents still exist below the high water mark, they are released as free storage. Next, the high water mark is moved to the page in the table space just after the last in-use extent. In table spaces where reclaimable storage is available, you use the ALTER TABLESPACE statement to reclaim unused extents. Figure 10 on page 137 shows a high-level view of how reclaimable storage works.

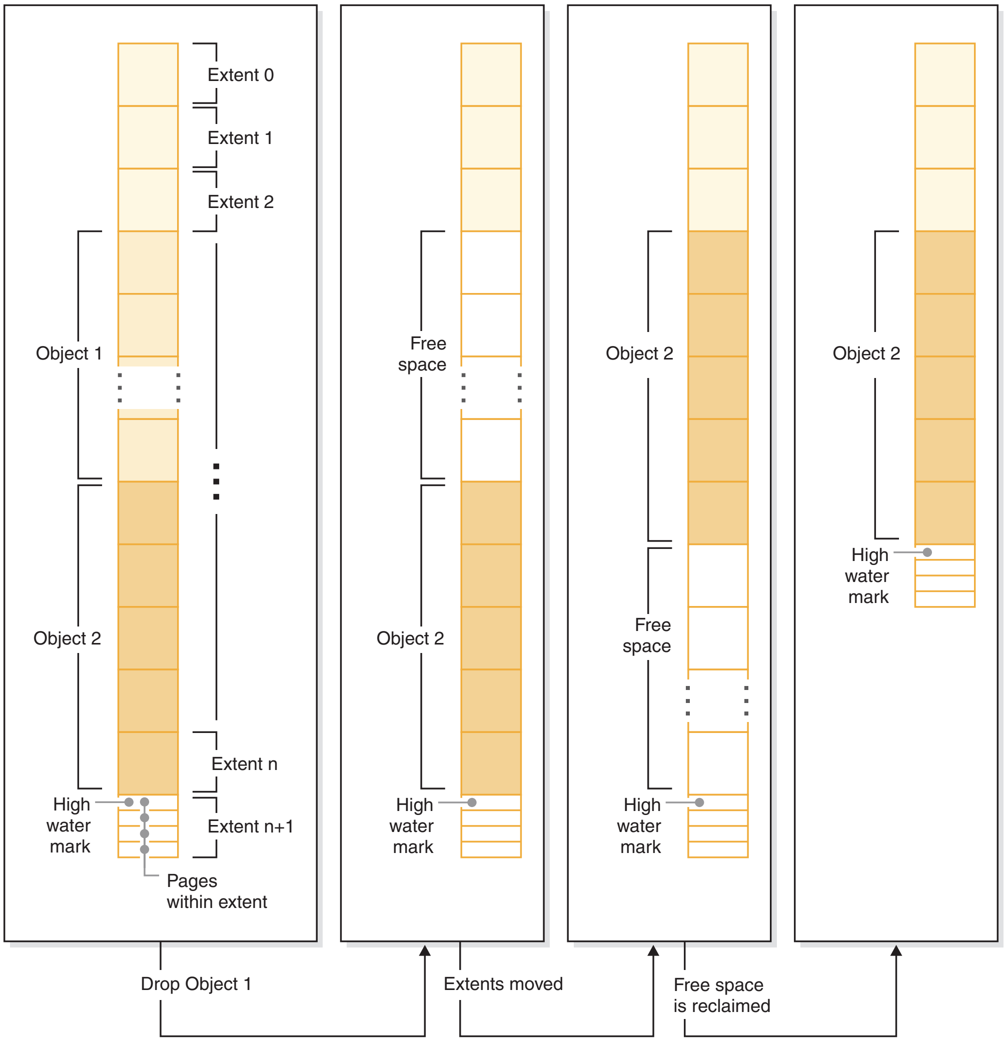


Figure 10. How reclaimable storage works. When reclaimable storage is enabled for a table space, the in-use extents can be moved to occupy unused extents lower in the table space.

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

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

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

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

Automatic storage table spaces

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

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

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

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

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

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

DMS table spaces

DMS table spaces can be reduced in two ways:

Container reduction only

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

Lower high water mark only

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

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

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

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

Examples

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

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

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

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

```
ALTER TABLESPACE TS REDUCE MAX
```

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

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

```
DROP TABLE T1
```

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

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

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

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

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

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

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

the database manager can lower the high water mark and reduce the container size without the need to perform any extent movement. This scenario is illustrated in Figure 11 on page 141

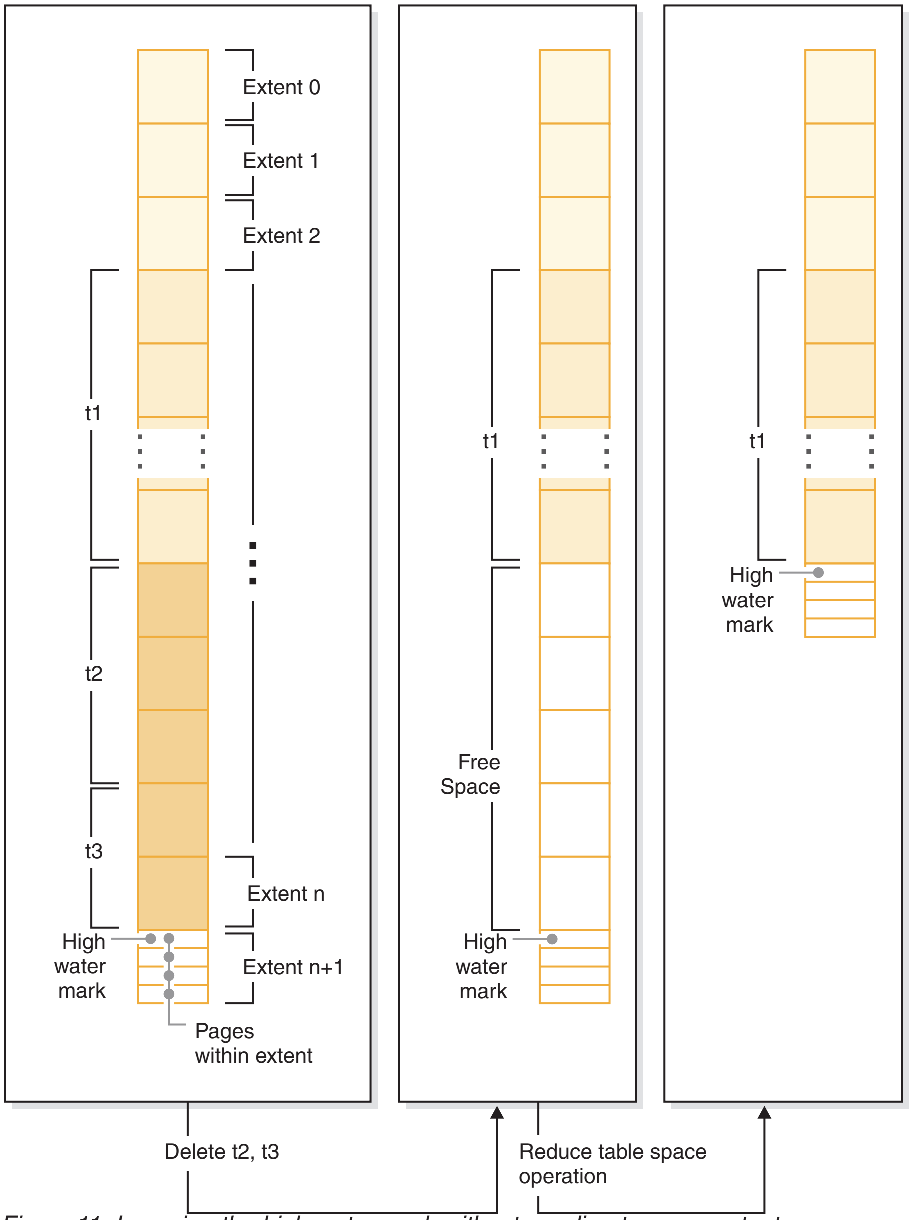


Figure 11. Lowering the high water mark without needing to move extents.

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

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

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

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

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

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

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

File system caching configurations

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

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

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

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

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

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

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

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

Supported configurations

Table 13 shows the supported configuration for using table spaces without file system caching. It also indicates: (a) whether DIO or enhanced DIO will be used in each case, and (b) the default behavior when neither NO FILE SYSTEM CACHING nor FILE SYSTEM CACHING is specified for a table space based on the platform and file system type.

Table 13. Supported configurations for table spaces without file system caching

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

Note:

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

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

Examples

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

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

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

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

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

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

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

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

Extent sizes in table spaces

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

When selecting an extent size, consider:

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

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

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

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

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

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

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

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

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

- The type of access to the tables.

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

- The minimum number of extents required.

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

Page, table and table space size

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

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

Table 14 shows the table space size limits for DMS and nontemporary automatic storage table spaces, by page size:

Table 14. Size limits for DMS and nontemporary automatic storage table spaces. DMS and nontemporary automatic storage table spaces are constrained by page size.

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

Table 14. Size limits for DMS and nontemporary automatic storage table spaces (continued). DMS and nontemporary automatic storage table spaces are constrained by page size.

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

Table 15 shows the table size limits tables in SMS and temporary automatic storage table spaces, by page size:

Table 15. Size limits for tables in SMS and temporary automatic storage table spaces. With tables in SMS and temporary automatic storage table spaces, it is the table objects themselves, not the table spaces that are constrained by page size.

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

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

Disk I/O efficiency and table space design

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

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

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

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

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

Whenever it is advantageous to do so, the database manager performs big-block reads. This typically occurs when retrieving data that is sequential or partially sequential in nature. The amount of data read in one read operation depends on the extent size - the bigger the extent size, the more pages can be read at one time.

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

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

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

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

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

For prefetching (or even reading) to be efficient, a sufficient number of clean buffer pool pages must exist. For example, there could be a parallel prefetch request that reads three extents from a table space, and for each page being read, one modified page is written out from the buffer pool. The prefetch request might be slowed down to the point where it cannot keep up with the query. Page cleaners should be configured in sufficient numbers to satisfy the prefetch request.

Table spaces in a partitioned database environment

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

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

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

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

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

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

Creating table spaces

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

About this task

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

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

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

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

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

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

Procedure

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

  ```
  CREATE TABLESPACE name
  ```

  or

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

  Assuming the table space is created in an automatic storage database, each of the two previously shown statements is equivalent; table spaces created in such a database will, by default, be automatic storage table spaces unless you specify otherwise.
- To create an SMS table space using the command line, enter:

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

  Important: The SMS table space type has been deprecated in Version 10.1 for user-defined permanent table spaces and might be removed in a future release. The SMS table space type is not deprecated for catalog and temporary table spaces. For more information, see "SMS permanent table spaces have been deprecated" in *What's New for DB2 Version 10.1*
- To create a DMS table space using the command line, enter:

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

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

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

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

Example

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

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

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

Example 2: Creating an SMS table space on Windows.

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

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

Example 3: Creating a DMS table space on Windows.

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

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

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

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

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

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

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

There are two methods to resolve this conflict:

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

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

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

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

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

```
IN database_partition_group_name
```

Example 4: Creating DMS table spaces on AIX.

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

```
CREATE TABLESPACE RESOURCE
      MANAGED BY DATABASE
      USING (DEVICE '/dev/rdblv6' 10000,
             DEVICE '/dev/rdblv7' 10000,
             DEVICE '/dev/rdblv8' 10000)
      OVERHEAD 7.5
      TRANSFERRATE 0.06
```

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

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

The following example creates a DMS table space on a database partition group called ODDGROUP in a UNIX multi-partition database. ODDGROUP must be previously created with a CREATE DATABASE PARTITION GROUP statement. In this case, the ODDGROUP database partition group is assumed to be made up of database partitions numbered 1, 3, and 5. On all database partitions, use the device /dev/hdisk0 for 10 000 4 KB pages. In addition, declare a device for each database partition of 40 000 4 KB pages.

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

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

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

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

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

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

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

Creating temporary table spaces

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

About this task

A *system temporary table space* is used to store system temporary tables. A database must always have at least one system temporary table space since system temporary tables can only be stored in such a table space. When a database is created, one of the three default table spaces defined is a system temporary table space called "TEMPSPACE1". You should have at least one system temporary table space of each page size for the user table spaces that exist in your database, otherwise some queries might fail. See “Table spaces for system, user and temporary data” on page 127 for more information.

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

Restrictions

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

Procedure

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

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

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

Defining initial table spaces on database creation

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

About this task

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

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

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

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

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

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

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

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

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

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

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

Altering automatic storage table spaces

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

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

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

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

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

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

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

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

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

Reclaiming unused storage in automatic storage table spaces

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

Before you begin

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

About this task

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

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

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

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

Note:

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

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

Restrictions

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

Procedure

To reduce the size of an automatic storage table space:

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

Example

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

```
ALTER TABLESPACE TS1 REDUCE MAX
```

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

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

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

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

Scenarios: Adding and removing storage with automatic storage table spaces

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

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

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

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

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

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

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

Figure 12 illustrates a storage path being added, with the "before" and "after" layout of a rebalanced table space:

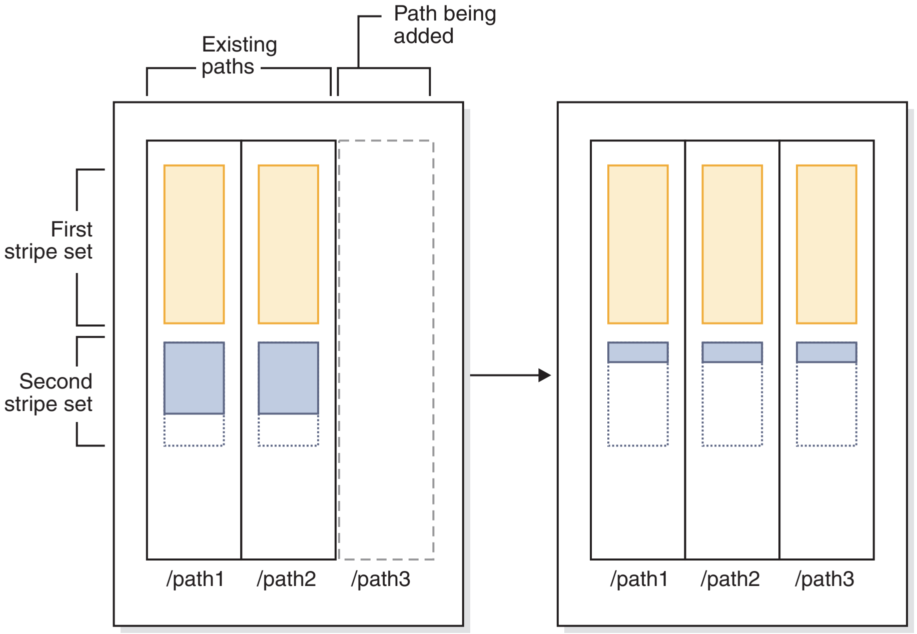


Figure 12. Adding a storage path and rebalancing an automatic storage table space

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

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

Figure 13 on page 159 is an example where a “hole” exists in the stripe sets of a table space (possibly caused by deleting table rows, for example) being rebalanced, with the “before” and “after” layout of the storage paths.

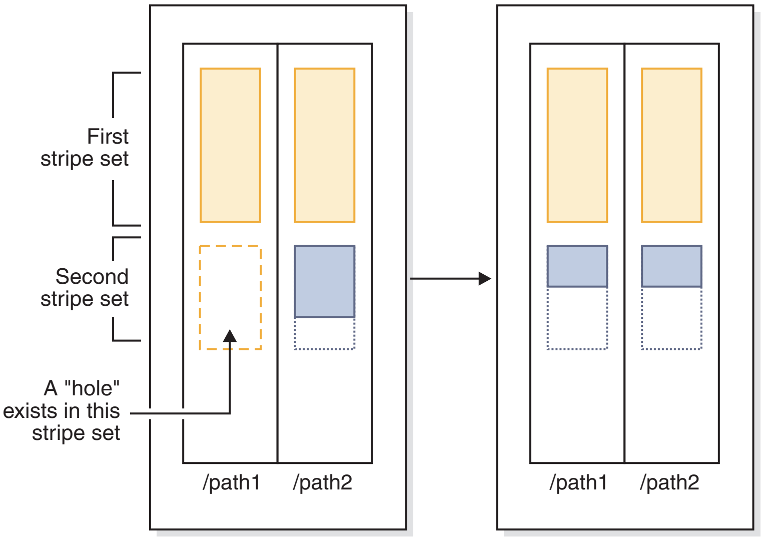


Figure 13. Rebalancing an automatic storage table space to fill gaps

Example

You created a storage group with two storage paths:

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

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

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

1. The first step is to add the storage path to the storage group:
   ```
   ALTER STOGROUP sg ADD '/path3'
   ```
2. The next step is to determine all of the affected permanent table spaces. This can be done by manually scanning table space snapshot output or via SQL. The following SQL statement will generate a list of all the regular and large automatic storage table spaces in the storage group:
   ```
   SELECT TBSP_NAME
      FROM table (MON_GET_TABLESPACE(' ', -2))
      WHERE TBSP_USING_AUTO_STORAGE = 1
         AND TBSP_CONTENT_TYPE IN ('ANY','LARGE')
         AND STORAGE_GROUP_NAME = 'sg'
      ORDER BY TBSP_ID
   ```
3. Once the table spaces have been identified, the next step is to perform the following statement for each of the table spaces listed. Provided that there is sufficient space on the remaining storage paths, it generally shouldn't matter what order the rebalances are performed in (and they can be run in parallel).
   ```
   ALTER TABLESPACE tablespace_name REBALANCE
   ```

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

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

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

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

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

When the REBALANCE operation is run:

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

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

Figure 14 on page 161 is an example of a storage path being dropped, with the “before” and “after” layout of the storage paths after the table space is rebalanced:

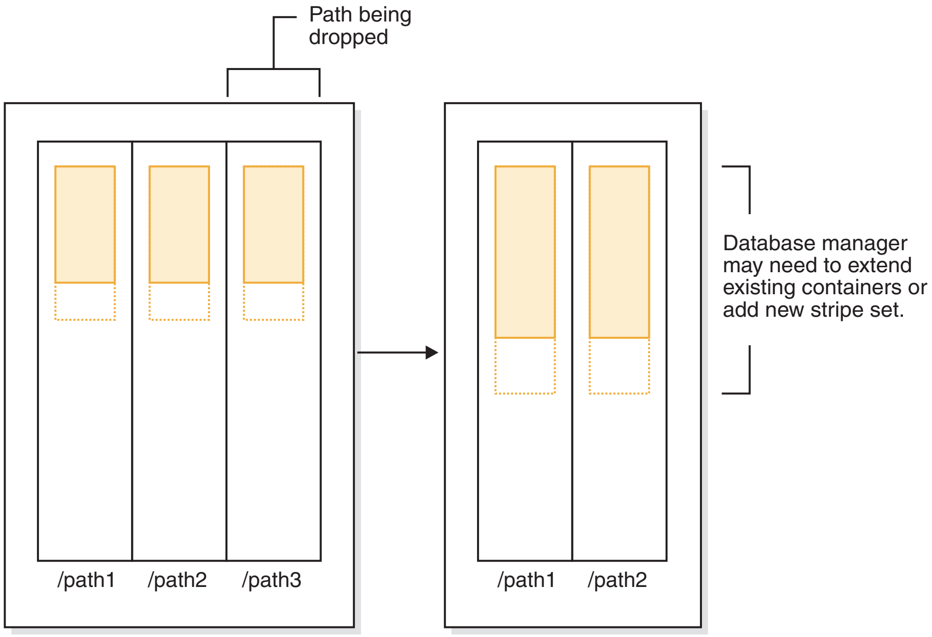

Path being dropped
Database manager may need to extend existing containers or add new stripe set.
/path1
/path2
/path3
/path1
/path2

Figure 14. Dropping a storage path and rebalancing an automatic storage table space

Example

Create a storage group with three storage paths:

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

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

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

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

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

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

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

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

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

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

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

b. You can then issue a REBALANCE statement for each table space in the result set.

4. To determine the list of affected table spaces, generate a list of all the system temporary and user temporary automatic storage table spaces that are defined on the dropped storage paths:

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

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

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

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

As described in “Scenario: Adding a storage path and rebalancing automatic storage table spaces” on page 157, a situation can occur in which the database manager fills in “holes” in stripe sets when dropping storage paths. In this case the database manager will create containers and drop containers as part of the process. In all of these scenarios, the database manager recognizes that some containers need to be added (where free space allows) and that some need to be removed. In these scenarios, the database manager might need to perform a two-pass rebalance operation (the phase and status of which is described in the snapshot monitor output):

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

Figure 15 on page 163 is an example of storage paths being added and dropped, with the "before" and "after" layout of a rebalanced table space:

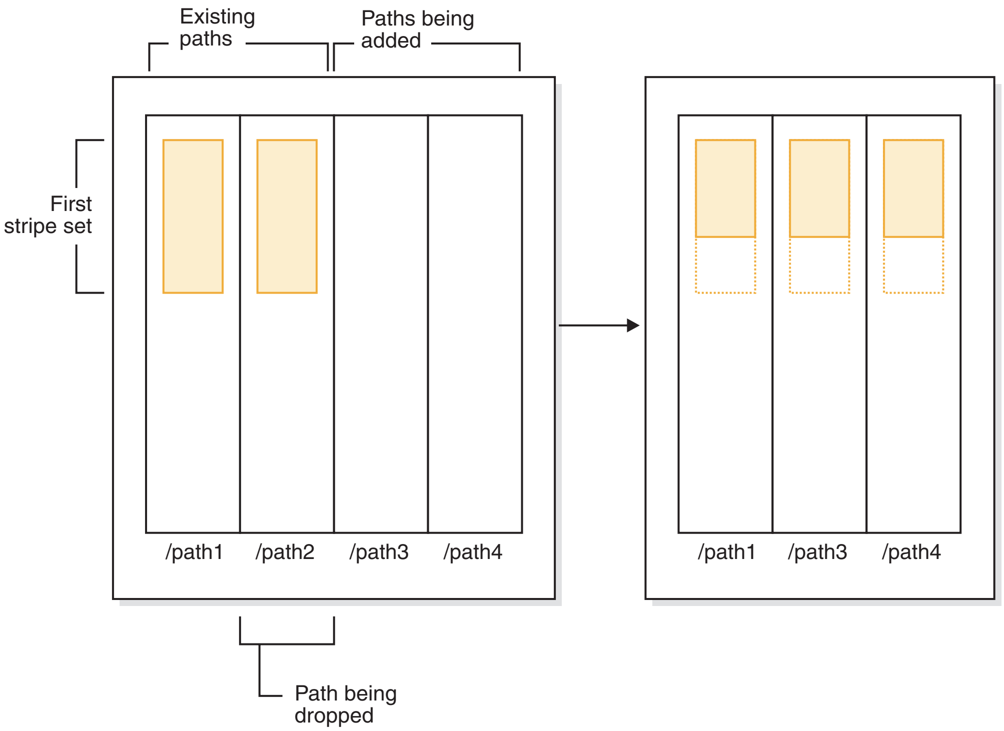


Figure 15. Adding and dropping storage paths, and then rebalancing an automatic storage table space

Example

A storage group is created with two storage paths:

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

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

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

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

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

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

```
ALTER TABLESPACE tablespace_name REBALANCE
```

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

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

Monitoring a table space rebalance operation

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

About this task

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

Procedure

To monitor a table space rebalance operation:

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

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

Results

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

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

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

  1 record(s) selected.
```

Table space states

This topic provides information about the supported table space states.

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

Table 16. Supported table space states

State	Hexadecimal state value	Description

Backup Pending	0x20	A table space is in this state after a point-in-time table space rollforward operation, or after a load operation (against a recoverable database) that specifies the **`COPY NO`** option. The table space (or, alternatively, the entire database) must be backed up before the table space can be used. If the table space is not backed up, tables within that table space can be queried, but not updated.
		Note: A database must also be backed up immediately after it is enabled for rollforward recovery. A database is recoverable if the **`logarchmeth1`** database configuration parameter is set to any value other than OFF. You cannot activate or connect to such a database until it has been backed up, at which time the value of the **`backup_pending`** informational database configuration parameter is set to NO.**Example**
		Given the `staff_data.del` input file with the following content:
		`11,"Melnyk",20,"Sales",10,70000,15000:`
		Load this data into the staff table specifying the copy no as follows:
		`update db cfg for sample using logarchmeth1 logretain;`
		`backup db sample;`
		`connect to sample;`
		`load from staff_data.del of del messages load.msg insert into staff copy no;`
		`update staff set salary = 69000 where id = 11;`
		`list tablespaces;`
		`connect reset;`
		Information returned for USERSPACE1 shows that this table space is in Backup Pending state.

Backup in Progress	0x800	This is a transient state that is only in effect during a backup operation.
		Example
		Perform an online backup as follows:
		`backup db sample online;`
		From another session, execute one of the following scripts while the backup operation is running:
		• `connect to sample;`
		`list tablespaces show detail;`
		`connect reset;`
		• `connect to sample;`
		`get snapshot for tablespaces on sample;`
		`connect reset;`
		Information returned for USERSPACE1 shows that this table space is in Backup in Progress state.

Table 16. Supported table space states (continued)

State	Hexadecimal state value	Description

DMS Rebalance in Progress	0x10000000	This is a transient state that is only in effect during a data rebalancing operation. When new containers are added to a table space that is defined as database managed space (DMS), or existing containers are extended, a rebalancing of the table space data might occur. *Rebalancing* is the process of moving table space extents from one location to another in an attempt to keep the data striped. An *extent* is a unit of container space (measured in pages), and a stripe is a layer of extents *across the set of containers* for a table space.
		Example
		Given the `staffdata.del` input file with 20000 or more records, create the table newstaff, load it using this input file, and then add a new container to table space ts1:
		`connect to sample;`
		`create tablespace ts1 managed by database using`
		` (file '/home/melnyk/melnyk/NODE0000/SQL00001/ts1c1' 1024);`
		`create table newstaff like staff in ts1;`
		`load from staffdata.del of del insert into newstaff nonrecoverable;`
		`alter tablespace ts1 add`
		` (file '/home/melnyk/melnyk/NODE0000/SQL00001/ts1c2' 1024);`
		`list tablespaces;`
		`connect reset;`
		Information returned for TS1 shows that this table space is in DMS Rebalance in Progress state.

Disable Pending	0x200	A table space may be in this state during a database rollforward operation and should no longer be in this state by the end of the rollforward operation. The state is triggered by conditions that result from a table space going offline and compensation log records for a transaction not being written. The appearance and subsequent disappearance of this table space state is transparent to users. An example illustrating this table space state is beyond the scope of this document.
Drop Pending	0x8000	A table space is in this state if one or more of its containers is found to have a problem during a database restart operation. (A database must be restarted if the previous session with this database terminated abnormally, such as during a power failure, for example.) If a table space is in Drop Pending state, it will not be available, and can only be dropped. An example illustrating this table space state is beyond the scope of this document.

Table 16. Supported table space states (continued)

State	Hexadecimal state value	Description

Load in Progress	0x20000	This is a transient state that is only in effect during a load operation (against a recoverable database) that specifies the **COPY NO** option. See also Load in Progress table state.
		Example
		Given the `staffdata.del` input file with 20000 or more records, create the table newstaff and load it specifying **COPY NO** and this input file:
		`update db cfg for sample using logarchmeth1 logretain;`
		`backup db sample;`
		`connect to sample;`
		`create table newstaff like staff;`
		`load from staffdata.del of del insert into newstaff copy no;`
		`connect reset;`
		From another session, get information about table spaces while the load operation is running by executing one of the sample scripts shown in the Backup in Progress example.
		Information returned for USERSPACE1 shows that this table space is in Load in Progress (and Backup Pending) state.

Normal	0x0	A table space is in Normal state if it is not in any of the other (abnormal) table space states. Normal state is the initial state of a table space after it is created.
		Example
		Create a table space and then get information about that table space as follows:
		`connect to sample;`
		`create tablespace ts1 managed by automatic storage;`
		`list tablespaces show detail;`
		Information returned for USERSPACE1 shows that this table space is in Normal state.

Table 16. Supported table space states (continued)

State	Hexadecimal state value	Description

Offline and Not Accessible	0x4000	A table space is in this state if there is a problem with one or more of its containers. A container might be inadvertently renamed, moved, or damaged. After the problem has been rectified, and the containers that are associated with the table space are accessible again, this abnormal state can be removed by disconnecting all applications from the database and then reconnecting to the database. Alternatively, you can issue an ALTER TABLESPACE statement, specifying the SWITCH ONLINE clause, to remove the Offline and Not Accessible state from the table space without disconnecting other applications from the database.
		Example
		Create table space ts1 with containers tsc1 and tsc2, create table staffemp, and import data from the `st_data.del` file as follows:
		`connect to sample;`
		`create tablespace ts1 managed by database using`
		` (file '/home/melnyk/melnyk/NODE0000/SQL00001/tsc1' 1024);`
		`alter tablespace ts1 add`
		` (file '/home/melnyk/melnyk/NODE0000/SQL00001/tsc2' 1024);`
		`export to st_data.del of del select * from staff;`
		`create table stafftemp like staff in ts1;`
		`import from st_data.del of del insert into stafftemp;`
		`connect reset;`
		Rename table space container tsc1 to tsc3 and then try to query the STAFFTEMP table:
		`connect to sample;`
		`select * from stafftemp;`
		The query returns SQL0290N (table space access is not allowed), and the **LIST TABLESPACES** command returns a state value of 0x4000 (Offline and Not Accessible) for TS1. Rename table space container tsc3 back to tsc1. This time the query runs successfully.

Quiesced Exclusive	0x4	A table space is in this state when the application that invokes the table space quiesce function has exclusive (read or write) access to the table space. Use the **QUIESCE TABLESPACES FOR TABLE** command to explicitly set a table space to Quiesced Exclusive.
		Example
		Set table spaces to Normal before setting them to Quiesced Exclusive as follows:
		`connect to sample;`
		`quiesce tablespaces for table staff reset;`
		`quiesce tablespaces for table staff exclusive;`
		`connect reset;`
		From another session, execute the following script:
		`connect to sample;`
		`select * from staff where id=60;`
		`update staff set salary=50000 where id=60;`
		`list tablespaces;`
		`connect reset;`
		Information returned for USERSPACE1 shows that this table space is in Quiesced Exclusive state.

Table 16. Supported table space states (continued)

State	Hexadecimal state value	Description

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

Quiesced Update	0x2	A table space is in this state when the application that invokes the table space quiesce function has exclusive write access to the table space. Use the **QUIESCE TABLESPACES FOR TABLE** command to explicitly set a table space to Quiesced Update state.
		Example
		Set table spaces to Normal before setting them to Quiesced Update as follows:
		`connect to sample;`
		`quiesce tablespaces for table staff reset;`
		`quiesce tablespaces for table staff intent to update;`
		`connect reset;`
		From another session, execute the following script:
		`connect to sample;`
		`select * from staff where id=50;`
		`update staff set salary=50000 where id=50;`
		`list tablespaces;`
		`connect reset;`
		Information returned for USERSPACE1 shows that this table space is in Quiesced Update state.

Table 16. Supported table space states (continued)

State	Hexadecimal state value	Description

Reorg in Progress	0x400	This is a transient state that is only in effect during a reorg operation.
		Example
		Reorganize the staff table as follows:
		`connect to sample;`
		`reorg table staff;`
		`connect reset;`
		From another session, get information about table spaces while the reorg operation is running by executing one of the sample scripts shown in the Backup in Progress example.
		Information returned for USERSPACE1 shows that this table space is in Reorg in Progress state.
		Note: Table reorganization operations involving the SAMPLE database are likely to complete in a short period of time and, as a result, it may be difficult to observe the Reorg in Progress state using this approach.

Restore Pending	0x100	Table spaces for a database are in this state after the first part of a redirected restore operation (that is, before the **SET TABLESPACE CONTAINERS** command is issued). The table space (or the entire database) must be restored before the table space can be used. You cannot connect to the database until the restore operation has been successfully completed, at which time the value of the **restore_pending** informational database configuration parameter is set to NO. **Example** When the first part of the redirected restore operation in Storage May be Defined completes, all of the table spaces are in Restore Pending state.

Restore in Progress	0x2000	This is a transient state that is only in effect during a restore operation.
		Example
		Enable the sample database for rollforward recovery then back up the sample database and the USERSPACE1 table space as follows:
		`update db cfg for sample using logarchmeth1 logretain;`
		`backup db sample;`
		`backup db sample tablespace (userspace1);`
		Restore the USERSPACE1 table space backup assuming the timestamp for this backup image is 20040611174124:
		`restore db sample tablespace (userspace1) online taken at 20040611174124;`
		From another session, get information about table spaces while the restore operation is running by executing one of the sample scripts shown in the Backup in Progress example.
		Information returned for USERSPACE1 shows that this table space is in Restore in Progress state.

Table 16. Supported table space states (continued)

State	Hexadecimal state value	Description
Roll Forward Pending	0x80	A table space is in this state after a restore operation against a recoverable database. The table space (or the entire database) must be rolled forward before the table space can be used. A database is recoverable if the **logarchmeth1** database configuration parameter is set to any value other than OFF. You cannot activate or connect to the database until a rollforward operation has been successfully completed, at which time the value of the **rollfwd_pending** informational database configuration parameter is set to NO. **Example** When the online table space restore operation in Restore in Progress completes, the table space USERSPACE1 is in Roll Forward Pending state.

Roll Forward in Progress	0x40	This is a transient state that is only in effect during a rollforward operation.
		Example
		Given the `staffdata.del` input file with 20000 or more record, create a table and tablespace followed by a database backup:
		`update db cfg for sample using logarchmeth1 logretain;`
		`backup db sample;`
		`connect to sample;`
		`create tablespace ts1 managed by automatic storage;`
		`create table newstaff like staff in ts1;`
		`connect reset;`
		`backup db sample tablespace (ts1) online;`
		Assuming that the timestamp for the backup image is `20040630000715`, restore the database backup and rollforward to the end of logs as follows:
		`connect to sample;`
		`load from staffdata.del of del insert into newstaff copy yes`
		` to /home/melnyk/backups;`
		`connect reset;`
		`restore db sample tablespace (ts1) online taken at 20040630000715;`
		`rollforward db sample to end of logs and stop tablespace (ts1) online;`
		From another session, get information about table spaces while the rollforward operation is running by executing one of the sample scripts shown in the Backup in Progress example.
		Information returned for TS1 shows that this table space is in Roll Forward in Progress state.

Storage May be Defined	0x2000000	Table spaces for a database are in this state after the first part of a redirected restore operation (that is, before the **SET TABLESPACE CONTAINERS** command is issued). This allows you to redefine the containers.
		Example
		Assuming that the timestamp for the backup image is `20040613204955`, restore a database backup as follows:
		`restore db sample taken at 20040613204955 redirect;`
		`list tablespaces;`
		Information returned by the **LIST TABLESPACES** command shows that all of the table spaces are in Storage May be Defined and Restore Pending state.

Table 16. Supported table space states (continued)

State	Hexadecimal state value	Description

Storage Must be Defined	0x1000	Table spaces for a database are in this state during a redirected restore operation to a new database if the set table space containers phase is omitted or if, during the set table space containers phase, the specified containers cannot be acquired. The latter can occur if, for example, an invalid path name has been specified, or there is insufficient disk space.
		Example
		Assuming that the timestamp for the backup image is `20040613204955`, restore a database backup as follows:
		`restore db sample taken at 20040613204955 into mydb redirect;`
		`set tablespace containers for 2 using (path 'ts2c1');`
		`list tablespaces;`
		Information returned by the **`LIST TABLESPACES`** command shows that table space SYSCATSPACE and table space TEMPSPACE1 are in Storage Must be Defined, Storage May be Defined, and Restore Pending state. Storage Must be Defined state takes precedence over Storage May be Defined state.

Suspend Write	0x10000	A table space is in this state after a write operation has been suspended. An example illustrating this table space state is beyond the scope of this document.

Table Space Creation in Progress	0x40000000	This is a transient state that is only in effect during a create table space operation.
		Example
		Create table spaces ts1, ts2, and ts3 as follows:
		`connect to sample;`
		`create tablespace ts1 managed by automatic storage;`
		`create tablespace ts2 managed by automatic storage;`
		`create tablespace ts3 managed by automatic storage;`
		From another session, get information about table spaces while the create table space operations are running by executing one of the sample scripts shown in the Backup in Progress example.
		Information returned for TS1, TS2, and TS3 shows that these table spaces are in Table Space Creation in Progress state.

Table Space Deletion in Progress	0x20000000	This is a transient state that is only in effect during a delete table space operation.
		Example
		Create table spaces ts1, ts2, and ts3 then drop them as follows:
		`connect to sample;`
		`create tablespace ts1 managed by automatic storage;`
		`create tablespace ts2 managed by automatic storage;`
		`create tablespace ts3 managed by automatic storage;`
		`drop tablespaces ts1, ts2, ts3;`
		From another session, get information about table spaces while the drop table space operations are running by executing one of the sample scripts shown in the Backup in Progress example.
		Information returned for TS1, TS2, and TS3 shows that these table spaces are in Table Space Deletion in Progress state.

Switching table spaces from offline to online

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

Procedure

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

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

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

Results

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

Dropping table spaces

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

About this task

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

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

Procedure

- Dropping user table spaces:

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

  The following SQL statement drops the table space ACCOUNTING:

  ```
  DROP TABLESPACE ACCOUNTING
  ```
- Dropping user temporary table spaces:

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

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

- Dropping system temporary table spaces:

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

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

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

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

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

     ```
     DROP TABLESPACE name
     ```

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

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

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

     ```
     DROP TABLESPACE TEMPSPACE1
     ```

Chapter 16. Storage groups

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

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

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

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

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

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

Data management using multi-temperature storage

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

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

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

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

The following example illustrates the use of storage groups with multi-temperature data. Assume that you are the DBA for a business that does most of its processing on current-fiscal-quarter data. As shown in Figure 16 on page 177, this business has enough solid-state drive (SSD) capacity to hold data for an entire quarter and enough Fibre Channel-based (FC) and Serial Attached SCSI (SAS) drive capacity to hold data for the remainder of the year. The data that is older then one year is stored on a large Serial ATA (SATA) RAID array that, while stable, does not perform quickly enough to withstand a heavy query workload. You can define three storage groups: one for the SSD storage (`sg_hot`), one for the FC and SAS storage (`sg_warm`), and the other for the SATA storage (`sg_cold`). You then take the following actions:

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

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

- Assign a table space for the new quarter to the sg_hot storage group
- Move the table space for the quarter that just passed to the sg_warm storage group
- Move the data for the oldest quarter in the sg_warm storage group to the sg_cold storage group

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

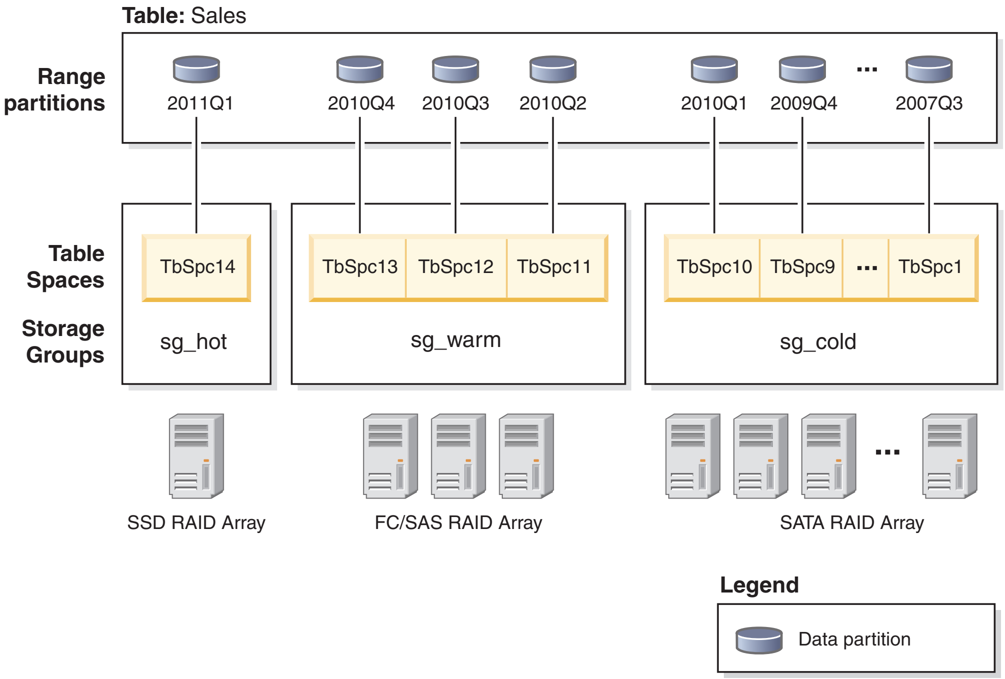


Figure 16. Managing Sales data using multi-temperature data storage

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

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

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

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

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

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

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

3. Set up your range partitions in your sales table.

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

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

4. After the current quarter passes, create a table space for a new quarter, and assign the table space to the sg_hot storage group.

   ```
   CREATE TABLESPACE tbsp_2011q2 USING STOGROUP sg_hot;
   ```

5. Move the table space for the quarter that just passed to the sg_warm storage group. To change the storage group association for the tbsp_2011q1 table space, issue the ALTER TABLESPACE statement with the USING STOGROUP option.

```
ALTER TABLESPACE tbsp_2011q1 USING STOGROUP sg_warm;
```

Default storage groups

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

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

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

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

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

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

Creating storage groups

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

Before you begin

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

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

Procedure

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

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

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

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

Altering storage groups

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

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

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

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

Adding storage paths

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

About this task

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

Procedure

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

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

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

  Important: All the paths that you assign to a storage group should have similar media characteristics: underlying disks, latency, device read rate, and size. If paths have non-uniform media characteristics, performance might be inconsistent.
- After adding one or more storage paths to the storage group, you can optionally use the ALTER TABLESPACE statement to rebalance table spaces to immediately start using the new storage paths. Otherwise, the new storage paths are used only when there is no space in the containers on the existing storage paths. To determine all of the affected permanent table spaces in the storage group, run the following statement:

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

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

```
ALTER TABLESPACE tablespace_name REBALANCE
```

where *tablespace_name* is the table space.

Dropping storage paths

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

Before you begin

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

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

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

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

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

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

Restrictions

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

About this task

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

Procedure

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

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

   where *sg* is the storage group and */db2/filesystem1* and */db2/filesystem2* are the storage paths being dropped.
2. Rebalance the containers of the storage paths being dropped. To determine all the affected permanent table spaces in the database that have containers residing on a "Drop Pending" path, issue the following statement:

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

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

   ```
   ALTER TABLESPACE tablespace_name REBALANCE
   ```

   where *tablespace_name* is a table space.

   After the last rebalance operation is complete, `/db2/filesystem1` and `/db2/filesystem2` are removed from the storage group.
3. Drop the temporary table spaces using the storage group. A table space in DROP_PENDING state is not dropped if there is a temporary table space on it.
4. Re-create the temporary table spaces that were using the storage group.

What to do next

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

Monitoring storage paths

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

The following administrative views and table functions can be used:

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

Replacing the paths of a storage group

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

Procedure

To replace the existing storage paths in a storage group:

1. Add the new storage paths to an existing storage group.
   ```
   ALTER STOGROUP sg_default ADD '/hdd/path3', '/hdd/path4'
   ```
2. Drop the old storage paths.
   ```
   ALTER STOGROUP sg_default DROP '/hdd/path1', '/hdd/path2'
   ```

   Note: All storage groups must have at least one path and that last path cannot be dropped.
   This marks the dropped storage paths as DROP PENDING.
3. Determine the affected non-temporary table spaces.
   ```
   SELECT TBSP_NAME
      FROM table (MON_GET_TABLESPACE(' ', -2))
      WHERE TBSP_USING_AUTO_STORAGE = 1
         AND TBSP_CONTENT_TYPE IN ('ANY','LARGE')
         AND STORAGE_GROUP_NAME = 'sg_default'
      ORDER BY TBSP_ID
   ```
4. Perform the following statement for each of the affected non-temporary table spaces returned.
   ```
   ALTER TABLESPACE tablespace-name REBALANCE
   ```
5. If there are any temporary table spaces defined on the dropped storage paths, you must create the new temporary table spaces first before dropping the old ones.
   ```
   SELECT TBSP_NAME
      FROM table (MON_GET_TABLESPACE(' ', -2))
      WHERE TBSP_USING_AUTO_STORAGE = 1
         AND TBSP_CONTENT_TYPE IN ('USRTEMP','SYSTEMP')
         AND STORAGE_GROUP_NAME = 'sg_default'
      ORDER BY TBSP_ID
   ```

Renaming storage groups

Use the RENAME STOGROUP statement to rename a storage group.

Procedure

Use the following statement to rename a storage group:

```
RENAME STOGROUP sg_hot TO sg_warm
```

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

Example

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

```
RENAME STOGROUP IBMSTOGROUP TO DEFAULT_SG
```

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

Dropping storage groups

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

About this task

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

Restrictions

You cannot drop the current default storage group.

Procedure

To drop a storage group:

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

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

   where *STO_GROUP* is the storage group that you want to drop.
2. If there are regular or large table spaces that use the storage group, assign them to a different storage group:

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

   where *sto_group_new* is a different storage group.
3. If there are temporary table spaces that use the storage group that you want to drop, perform these steps:
   a. Determine what temporary table spaces use the storage group that you want to drop:

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

   b. Drop the temporary table spaces using the storage group:

      ```
      DROP TABLESPACE table_space
      ```

   c. Re-create the temporary table spaces that were using the storage group.
4. Monitor the rebalance activity for the storage group to be dropped.

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

   An empty result state indicates that all table spaces have finished moving to the new storage group.
5. Drop the storage group when all table space extents have been successfully moved to the target storage group.

   ```
   DROP STOGROUP STO_GROUP
   ```

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

Storage group and table space media attributes

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

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

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

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

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

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

Table 17. The default settings for storage group attributes

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

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

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

```
TRANSFERRATE = ( 1 / DEVICE READ RATE ) * 1000 / 1024000 * PAGESIZE
```

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

Table 18. Default TRANSFERRATE values

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

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

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

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

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

Associating a table space to a storage group

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

About this task

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

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

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

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

Procedure

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

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

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

Scenario: Moving a table space to a new storage group

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

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

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

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

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

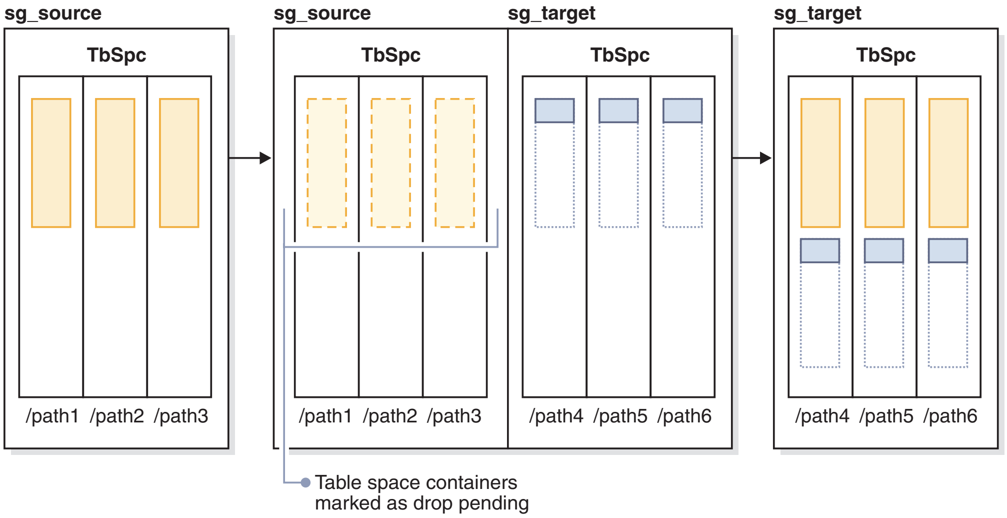


Figure 17. Moving a table space to a new storage group

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

1. Create two storage groups, sg_source and sg_target:

   ```
   CREATE STOGROUP sg_source ON '/path1', '/path2', '/path3'
   CREATE STOGROUP sg_target ON '/path4', '/path5', '/path6'
   ```

2. After creating the database, create an automatic storage table space that initially uses the sg_source storage group:

   ```
   CREATE TABLESPACE TbSpc USING STOGROUP sg_source
   ```

3. Move the automatic storage table space to the sg_target storage group:

   ```
   ALTER TABLESPACE TbSpc USING sg_target
   ```

Chapter 17. Schemas

A *schema* is a collection of named objects; it provides a way to group those objects logically. A schema is also a name qualifier; it provides a way to use the same natural name for several objects, and to prevent ambiguous references to those objects.

For example, the schema names 'INTERNAL' and 'EXTERNAL' make it easy to distinguish two different SALES tables (INTERNAL.SALES, EXTERNAL.SALES).

Schemas also enable multiple applications to store data in a single database without encountering namespace collisions.

A schema is distinct from, and should not be confused with, an *XML schema*, which is a standard that describes the structure and validates the content of XML documents.

A schema can contain tables, views, nicknames, triggers, functions, packages, and other objects. A schema is itself a database object. It is explicitly created using the CREATE SCHEMA statement, with the current user or a specified authorization ID recorded as the schema owner. It can also be implicitly created when another object is created, if the user has IMPLICIT_SCHEMA authority.

A *schema name* is used as the high order part of a two-part object name. If the object is specifically qualified with a schema name when created, the object is assigned to that schema. If no schema name is specified when the object is created, the default schema name is used (specified in the CURRENT SCHEMA special register).

For example, a user with DBADM authority creates a schema called C for user A:

```
CREATE SCHEMA C AUTHORIZATION A
```

User A can then issue the following statement to create a table called X in schema C (provided that user A has the CREATETAB database authority):

```
CREATE TABLE C.X (COL1 INT)
```

Some schema names are reserved. For example, built-in functions belong to the SYSIBM schema, and the pre-installed user-defined functions belong to the SYSFUN schema.

When a database is created, if it is not created with the `RESTRICTIVE` option, all users have IMPLICIT_SCHEMA authority. With this authority, users implicitly create a schema whenever they create an object with a schema name that does not already exist. When schemas are implicitly created, CREATEIN privileges are granted which allows any user to create other objects in this schema. The ability to create objects such as aliases, distinct types, functions, and triggers is extended to implicitly created schemas. The default privileges on an implicitly created schema provide backward compatibility with previous versions.

The owner of an implicitly created schema is SYSIBM. When the database is restrictive, PUBLIC does not have the CREATEIN privilege on the schema. The user who implicitly creates the schema has CREATEIN privilege on the schema. When the database is not restrictive, PUBLIC has the CREATEIN privilege on the schema.

If IMPLICIT_SCHEMA authority is revoked from PUBLIC, schemas can be explicitly created using the CREATE SCHEMA statement, or implicitly created by users (such as those with DBADM authority) who have been granted IMPLICIT_SCHEMA authority. Although revoking IMPLICIT_SCHEMA authority from PUBLIC increases control over the use of schema names, it can result in authorization errors when existing applications attempt to create objects.

Schemas also have privileges, allowing the schema owner to control which users have the privilege to create, alter, and drop objects in the schema. This ability provides a way to control the manipulation of a subset of objects in the database. A schema owner is initially given all of these privileges on the schema, with the ability to grant the privileges to others. An implicitly created schema is owned by the system, and all users are initially given the privilege to create objects in such a schema, except in a restrictive database environment. A user with ACCESSCTRL or SECADM authority can change the privileges that are held by users on any schema. Therefore, access to create, alter, and drop objects in any schema (even one that was implicitly created) can be controlled.

Designing schemas

when organizing your data into tables, it might be beneficial to group the tables and other related objects together. This is done by defining a schema through the use of the CREATE SCHEMA statement.

Information about the schema is kept in the system catalog tables of the database to which you are connected. As other objects are created, they can be placed within the schemas you create, however, note that an object can exist in only one schema.

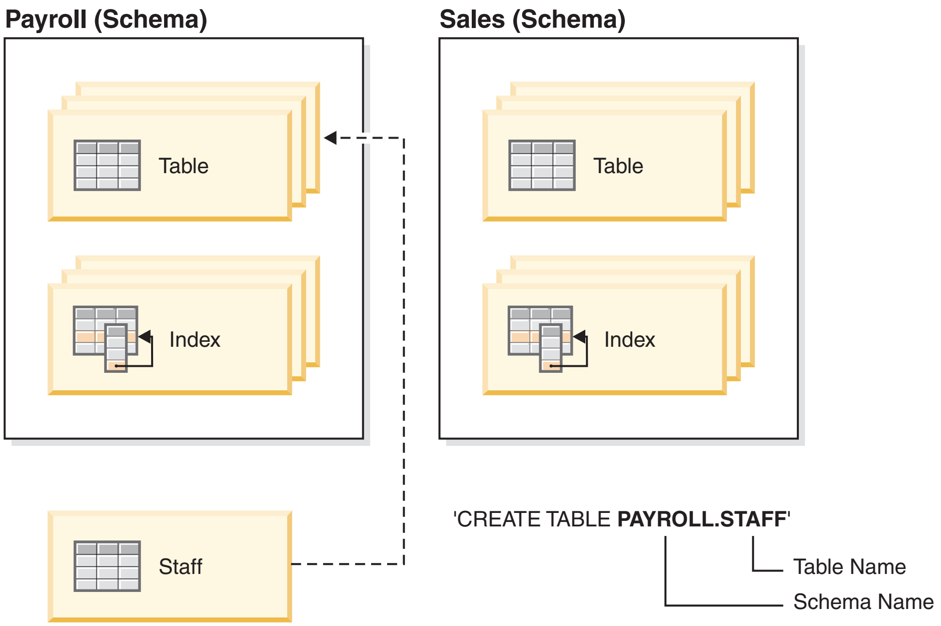
Schemas can be compared to directories, with the current schema being the current directory. Using this analogy, SET SCHEMA is equivalent to the **`change directory`** command.

Important: It is important to understand that there is no relation between authorization IDs and schemas except for the default CURRENT SCHEMA setting (described in the following section).

when designing your databases and tables, you should also consider the schemas in your system, including their names and the objects that will be associated with each of them.

Most objects in a database are assigned a unique name that consists of two parts. The first (leftmost) part is called the qualifier or schema, and the second (rightmost) part is called the simple (or unqualified) name. Syntactically, these two parts are concatenated as a single string of characters separated by a period. When any object that can be qualified by a schema name (such as a table, index, view, user-defined data type, user-defined function, nickname, package, or trigger) is first created, it is assigned to a particular schema based on the qualifier in its name.

For example, the following diagram illustrates how a table is assigned to a particular schema during the table creation process:



You should also be familiar with how schema access is granted, in order to give your users the correct authority and instructions:

Schema names

When creating a new schema, the name must not identify a schema name already described in the catalog and the name cannot begin with "SYS". For other restrictions and recommendations, see “Schema name restrictions and recommendations” on page 193.

Access to schemas

Unqualified access to objects within a schema is not allowed since the schema is used to enforce uniqueness in the database. This becomes clear when considering the possibility that two users could create two tables (or other objects) with the same name. Without a schema to enforce uniqueness, ambiguity would exist if a third user attempted to query the table. It is not possible to determine which table to use without some further qualification.

The definer of any objects created as part of the CREATE SCHEMA statement is the schema owner. This owner can GRANT and REVOKE schema privileges to other users.

If a user has DBADM authority, then that user can create a schema with any valid name. When a database is created, IMPLICIT_SCHEMA authority is granted to PUBLIC (that is, to all users).

If users do not have IMPLICIT_SCHEMA or DBADM authority, the only schema they can create is one that has the same name as their own authorization ID.

Default schema

If a schema or qualifier is not specified as part of the name of the object to be created, that object is assigned to the default schema as indicated in the CURRENT SCHEMA special register. The default value of this special register is the value of the session authorization ID.

A default schema is needed by unqualified object references in dynamic statements. You can set a default schema for a specific DB2 connection by setting the CURRENT SCHEMA special register to the schema that you want as the default. No designated authorization is required to set this special register, so any user can set the CURRENT SCHEMA.

The syntax of the SET SCHEMA statement is:

```
SET SCHEMA = <schema-name>
```

You can issue this statement interactively or from within an application. The initial value of the CURRENT SCHEMA special register is equal to the authorization ID of the current session user. For more information, see the SET SCHEMA statement.

Note:

- There are other ways to set the default schema upon connection. For example, by using the `cli.ini` file for CLI/ODBC applications, or by using the connection properties for the JDBC application programming interface.
- The default schema record is not created in the system catalogs, but it exists only as a value that the database manager can obtain (from the CURRENT SCHEMA special register) whenever a schema or qualifier is not specified as part of the name of the object to be created.

Implicit creation

You can implicitly create schemas if you have IMPLICIT_SCHEMA authority. With this authority, you can implicitly create a schema whenever you create an object with a schema name that does not already exist. Often schemas are implicitly created the first time a data object in the schema is created, provided the user creating the object holds the IMPLICIT_SCHEMA authority.

Explicit creation

Schemas can also be explicitly created and dropped by executing the CREATE SCHEMA and DROP SCHEMA statements from the command line or from an application program. For more information, see the CREATE SCHEMA and DROP SCHEMA statements.

Table and view aliases by schema

To allow another user to access a table or view without entering the schema name as part of the qualification on the table or view name requires that a an alias be established for that user. The definition of the alias would define the fully-qualified table or view name including the user's schema; then the user queries using the alias name. The alias would be fully-qualified by the user's schema as part of the alias definition.

Grouping objects by schema

Database object names might be made up of a single identifier or they might be *schema-qualified objects* made up of two identifiers. The schema, or high-order part, of a schema-qualified object provides a means to classify or group objects in the database. When an object such as a table, view, alias, distinct type, function, index, package or trigger is created, it is assigned to a schema. This assignment is done either explicitly or implicitly.

Explicit use of the schema occurs when you use the high-order part of a two-part object name when referring to that object in a statement. For example, USER A issues a CREATE TABLE statement in schema C as follows:

```
CREATE TABLE C.X (COL1 INT)
```

Implicit use of the schema occurs when you do not use the high-order part of a two-part object name. When this happens, the CURRENT SCHEMA special register is used to identify the schema name used to complete the high-order part of the object name. The initial value of CURRENT SCHEMA is the authorization ID of the current session user. If you want to change this during the current session, you can use the SET SCHEMA statement to set the special register to another schema name.

Some objects are created within certain schemas and stored in the system catalog tables when the database is created.

You do not have to explicitly specify in which schema an object is to be created; if not specified, the authorization ID of the statement is used. For example, for the following CREATE TABLE statement, the schema name defaults to the authorization ID that is currently logged on (that is, the CURRENT SCHEMA special register value):

```
CREATE TABLE X (COL1 INT)
```

Dynamic SQL and XQuery statements typically use the CURRENT SCHEMA special register value to implicitly qualify any unqualified object name references.

Before creating your own objects, you must consider whether you want to create them in your own schema or by using a different schema that logically groups the objects. If you are creating objects that will be shared, using a different schema name can be very beneficial.

Schema name restrictions and recommendations

There are some restrictions and recommendations that you must be aware of when naming schemas.

- User-defined types (UDTs) cannot have schema names longer than the schema length listed in “SQL and XML limits” in the *SQL Reference*.
- The following schema names are reserved words and must not be used: SYSCAT, SYSFUN, SYSIBM, SYSSTAT, SYSPROC.
- To avoid potential problems upgrading databases in the future, do not use schema names that begin with SYS. The database manager will not allow you to create modules, procedures, triggers, user-defined types or user-defined functions using a schema name beginning with SYS.
- It is recommended that you not use SESSION as a schema name. Declared temporary tables must be qualified by SESSION. It is therefore possible to have an application declare a temporary table with a name identical to that of a persistent table, in which case the application logic can become overly complicated. Avoid the use of the schema SESSION, except when dealing with declared temporary tables.

Creating schemas

You can use schemas to group objects as you create those objects. An object can belong to only one schema. Use the CREATE SCHEMA statement to create schemas.

Information about the schemas is kept in the system catalog tables of the database to which you are connected.

Before you begin

To create a schema and optionally make another user the owner of the schema, you need DBADM authority. If you do not hold DBADM authority, you can still create a schema using your own authorization ID. The definer of any objects created as part of the CREATE SCHEMA statement is the schema owner. This owner can GRANT and REVOKE schema privileges to other users.

Procedure

To create a schema from the command line, enter the following statement:

```
CREATE SCHEMA schema-name [ AUTHORIZATION schema-owner-name ]
```

Where *schema-name* is the name of the schema. This name must be unique within the schemas already recorded in the catalog, and the name cannot begin with SYS. If the optional AUTHORIZATION clause is specified, the *schema-owner-name* becomes the owner of the schema. If this clause is not specified, the authorization ID that issued this statement becomes the owner of the schema.
For more information, see the CREATE SCHEMA statement. See also “Schema name restrictions and recommendations” on page 193.

Dropping schemas

To delete a schema, use the DROP statement.

Before you begin

Before dropping a schema, all objects that were in that schema must be dropped or moved to another schema.

The schema name must be in the catalog when attempting the DROP statement; otherwise an error is returned.

Procedure

To drop a schema by using the command line, enter:

```
DROP SCHEMA name RESTRICT
```

The RESTRICT keyword enforces the rule that no objects can be defined in the specified schema for the schema to be deleted from the database. The RESTRICT keyword is not optional.

Example

In the following example, the schema "joeschma" is dropped:

```
DROP SCHEMA joeschma RESTRICT
```

Chapter 18. Database objects

Soft invalidation of database objects

When *soft invalidation* is active, an object can be dropped even if other running transactions are using it. Transactions that were using the dropped object are permitted to continue, but any new transaction will be denied access to the dropped object.

All cached statements and packages that directly or indirectly refer to the object being dropped or altered are marked as not valid (and are said to be *invalidated*). Soft invalidation allows DDL affecting the referenced objects to avoid waits that otherwise would result from statements being run holding locks on objects to which they refer, and allows any active access to continue using a cached version of the object, eliminating the possibility of lock timeouts.

By contrast, when *hard invalidation* is used, exclusive locking is used when referencing an object. This guarantees that all processes are using the same versions of objects and that there are no accesses to an object once it has been dropped.

Soft invalidation is enabled through the **DB2_DDL_SOFT_INVAL** registry variable; by default, this registry variable is set to ON.

The following list shows the data definition language (DDL) statements for which soft invalidation is supported:

- ALTER TABLE...DETACH PARTITION
- CREATE OR REPLACE ALIAS
- CREATE OR REPLACE FUNCTION
- CREATE OR REPLACE TRIGGER
- CREATE OR REPLACE VIEW
- DROP ALIAS
- DROP FUNCTION
- DROP TRIGGER
- DROP VIEW

Note: In DB2 Version 9.7 Fix Pack 1 and later releases, ALTER TABLE...DETACH PARTITION performs soft invalidation at all isolation levels on cached statements that directly or indirectly refer to the partitioned table. A subsequent asynchronous partition detach task performs hard invalidation on previously soft invalidated cached statements before converting the detached partition into a stand-alone table.

The **DB2_DDL_SOFT_INVAL** registry variable does not affect the invalidation done by ALTER TABLE...DETACH PARTITION.

Soft invalidation support applies only to dynamic SQL and to scans done under the cursor stability (CS) and uncommitted read (UR) isolation levels. For the ALTER TABLE...DETACH PARTITION statement, the soft invalidation applies to scans under all isolation levels.

Example

Assume a view called VIEW1 exists. You open a cursor, and run the statement `SELECT * from VIEW1`. Shortly afterward, the database administrator issues the command `DROP VIEW VIEW1` to drop VIEW1 from the database. With hard invalidation, the DROP VIEW statement will be forced to wait for an exclusive lock on VIEW1 until the SELECT transaction has finished. With soft invalidation, the DROP VIEW statement is not given an exclusive lock on the view. The view is dropped, however, the SELECT statement will continue to run using the most recent definition of the view. Once the SELECT statement has completed, any subsequent attempts to use to VIEW1 (even by the same user or process that just used it) will result in an error (SQL0204N).

Automatic revalidation of database objects

Automatic revalidation is a mechanism whereby invalid database objects are automatically revalidated when accessed at run time.

A database object usually depends upon one or more different base objects. If the status of base objects on which the database object depends upon change in any important way, such as the base object being altered or dropped, the dependent database object becomes invalid. Invalid database objects must be revalidated before they can be used again. Revalidation is the process by which the DB2 software reprocesses the definition of an invalid dependent object so that the object is updated with the current state of its base objects, thereby turning the invalid dependent object back into a usable, valid object. Automatic revalidation is a mechanism whereby invalid database objects are automatically revalidated when accessed at run time.

In general, the database manager attempts to revalidate invalid objects the next time that those objects are used. Automatic revalidation is enabled through the **`auto_reval`** configuration parameter. By default, this registry variable is set to DEFERRED, except for databases upgraded from Version 9.5 or earlier, in which case **`auto_reval`** is set to DISABLED.

For information about the dependent objects that are impacted when an object is dropped, and when those dependent objects are revalidated, see “DROP statement” in the *SQL Reference Volume 1*.

The following list shows the data definition language (DDL) statements for which automatic revalidation is currently supported:
- ALTER MODULE DROP FUNCTION
- ALTER MODULE DROP PROCEDURE
- ALTER MODULE DROP TYPE
- ALTER MODULE DROP VARIABLE
- ALTER NICKNAME (altering the local name or the local type)
- ALTER TABLE ALTER COLUMN
- ALTER TABLE DROP COLUMN
- ALTER TABLE RENAME COLUMN
- CREATE OR REPLACE ALIAS
- CREATE OR REPLACE FUNCTION
- CREATE OR REPLACE NICKNAME
- CREATE OR REPLACE PROCEDURE

- CREATE OR REPLACE SEQUENCE
- CREATE OR REPLACE TRIGGER
- CREATE OR REPLACE VARIABLE
- CREATE OR REPLACE VIEW
- DROP FUNCTION
- DROP NICKNAME
- DROP PROCEDURE
- DROP SEQUENCE
- DROP TABLE
- DROP TRIGGER
- DROP TYPE
- DROP VARIABLE
- DROP VIEW
- RENAME TABLE

You can use the ADMIN_REVALIDATE_DB_OBJECTS procedure to revalidate existing objects that have been marked invalid.

Creating database object aliases

An *alias* is an indirect method of referencing a table, nickname, or view, so that an SQL or XQuery statement can be independent of the qualified name of that table or view.

About this task

Only the alias definition must be changed if the table or view name changes. An alias can be created on another alias. An alias can be used in a view or trigger definition and in any SQL or XQuery statement, except for table check-constraint definitions, in which an existing table or view name can be referenced.

An alias can be defined for a table, view, or alias that does not exist at the time of definition. However, it must exist when the SQL or XQuery statement containing the alias is compiled.

An alias name can be used wherever an existing table name can be used, and can refer to another alias if no circular or repetitive references are made along the chain of aliases.

The alias name cannot be the same as an existing table, view, or alias, and can only refer to a table within the same database. The name of a table or view used in a CREATE TABLE or CREATE VIEW statement cannot be the same as an alias name in the same schema.

You do not require special authority to create an alias, unless the alias is in a schema other than the one owned by your current authorization ID, in which case DBADM authority is required.

When an alias, or the object to which an alias refers, is dropped, all packages dependent on the alias are marked as being not valid and all views and triggers dependent on the alias are marked inoperative.

Note: DB2 for z/OS employs two distinct concepts of aliases: ALIAS and SYNONYM. These two concepts differ from DB2 for Linux, UNIX, and Windows as follows:

- ALIASes in DB2 for z/OS:
  - Require their creator to have special authority or privilege
  - Cannot reference other aliases
- SYNONYMs in DB2 for z/OS:
  - Can only be used by their creator
  - Are always unqualified
  - Are dropped when a referenced table is dropped
  - Do not share namespace with tables or views

Procedure

To create an alias using the command line, enter:

```
CREATE ALIAS alias_name FOR table_name
```

The following SQL statement creates an alias WORKERS for the EMPLOYEE table:

```
CREATE ALIAS WORKERS FOR EMPLOYEE
```

The alias is replaced at statement compilation time by the table or view name. If the alias or alias chain cannot be resolved to a table or view name, an error results. For example, if WORKERS is an alias for EMPLOYEE, then at compilation time:

```
SELECT * FROM WORKERS
```

becomes in effect

```
SELECT * FROM EMPLOYEE
```

Creating and maintaining database objects

When creating some types of database objects, you should be aware of the CREATE with errors support, as well as the REPLACE option.

CREATE with errors support for certain database objects

Some types of objects can be created even if errors occur during their compilation; for example, creating a view when the table to which it refers does not exist.

Such objects remain invalid until they are accessed. CREATE with errors support currently extends to views and inline SQL functions (not compiled functions). This feature is enabled if the **auto_reval** database configuration parameter is set to IMMEDIATE or DEFERRED.

The errors that are tolerated during object creation are limited to the following types:

- Any name resolution error, such as: a referenced table does not exist (SQLSTATE 42704, SQL0204N), a referenced column does not exist (SQLSTATE 42703, SQL0206N), or a referenced function cannot be found (SQLSTATE 42884, SQL0440N)
- Any nested revalidation failure. An object being created can reference objects that are not valid , and revalidation will be invoked for those invalid objects. If revalidation of any referenced invalid object fails, the CREATE statement succeeds, and the created object will remain invalid until it is next accessed.

- Any authorization error (SQLSTATE 42501, SQL0551N)

An object can be created successfully even if there are multiple errors in its body. The warning message that is returned contains the name of the first undefined, invalid, or unauthorized object that was encountered during compilation. The SYSCAT.INVALIDOBJECTS catalog view contains information about invalid objects.

You can use the ADMIN_REVALIDATE_DB_OBJECTS procedure to revalidate existing objects that have been marked invalid.

Example

```
create view v2 as select * from v1
```

If v1 does not exist, the CREATE VIEW statement completes successfully, but v2 remains invalid.

REPLACE option on several CREATE statements

The **OR REPLACE** clause on the CREATE statement for several objects, including aliases, functions, modules, nicknames, procedures (including federated procedures), sequences, triggers, variables, and views allows the object to be replaced if it already exists; otherwise, it is created. This significantly reduces the effort required to change a database schema.

Privileges that were previously granted on an object are preserved when that object is replaced. In other respects, CREATE OR REPLACE is semantically similar to DROP followed by CREATE. In the case of functions, procedures, and triggers, support applies to both inline objects and compiled objects.

In the case of functions and procedures, support applies to both SQL and external functions and procedures. If a module is replaced, all the objects within the module are dropped; the new version of the module contains no objects.

Objects that depend (either directly or indirectly) on an object that is being replaced are invalidated. Revalidation of all dependent objects following a replace operation is always done immediately after the invalidation, even if the **auto_reval** database configuration parameter is set to DISABLED.

Example

Replace v1, a view that has dependent objects.

```
create table t1 (c1 int, c2 int);
create table t2 (c1 int, c2 int);

create view v1 as select * from t1;
create view v2 as select * from v1;

create function foo1()
 language sql
 returns int
 return select c1 from v2;

create or replace v1 as select * from t2;

select * from v2;

values foo1();
```

The replaced version of v1 references t2 instead of t1. Both v2 and foo1 are invalidated by the CREATE OR REPLACE statement. Under revalidation deferred semantics, `select * from v2` successfully revalidates v2, but not foo1, which is revalidated by values foo1(). Under revalidation immediate semantics, both v2 and foo1 are successfully revalidated by the CREATE OR REPLACE statement.

Chapter 19. Tables

Tables are logical structures maintained by the database manager. Tables are made up of columns and rows.

At the intersection of every column and row is a specific data item called a *value*. A *column* is a set of values of the same type or one of its subtypes. A *row* is a sequence of values arranged so that the *n*th value is a value of the *n*th column of the table.

An application program can determine the order in which the rows are populated into the table, but the actual order of rows is determined by the database manager, and typically cannot be controlled. Multidimensional clustering (MDC) provides some sense of clustering, but not actual ordering between the rows.

Types of tables

DB2 databases store data in tables. In addition to tables used to store persistent data, there are also tables that are used for presenting results, summary tables and temporary tables; multidimensional clustering tables offer specific advantages in a warehouse environment.

Base tables

These types of tables hold persistent data. There are different kinds of base tables, including

Regular tables

Regular tables with indexes are the "general purpose" table choice.

Multidimensional clustering (MDC) tables

These types of tables are implemented as tables that are physically clustered on more than one key, or dimension, at the same time. MDC tables are used in data warehousing and large database environments. Clustering indexes on regular tables support single-dimensional clustering of data. MDC tables provide the benefits of data clustering across more than one dimension. MDC tables provide *guaranteed clustering* within the composite dimensions. By contrast, although you can have a clustered index with regular tables, clustering in this case is attempted by the database manager, but not guaranteed and it typically degrades over time. MDC tables can coexist with partitioned tables and can themselves be partitioned tables.

Multidimensional clustering tables are not supported in a DB2 pureScale environment.

Insert time clustering (ITC) tables

These types of tables are conceptually, and physically similar to MDC tables, but rather than being clustered by one or more user specified dimensions, rows are clustered by the time they are inserted into the table. ITC tables can be partitioned tables.

ITC tables are not supported in a DB2 pureScale environment.

Range-clustered tables (RCT)

These types of tables are implemented as sequential clusters of data that provide fast, direct access. Each record in the table has a predetermined record ID (RID) which is an internal identifier used to locate a record in a table. RCT tables are used where the data is tightly clustered across one or more columns in the table. The largest and smallest values in the columns define the range of possible values. You use these columns to access records in the table; this is the most optimal method of using the predetermined record identifier (RID) aspect of RCT tables.

Range-clustered tables are not supported in a DB2 pureScale environment.

Partitioned tables

These types of tables use a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table. Data partitions can be added to, attached to, and detached from a partitioned table, and you can store multiple data partition ranges from a table in one table space. Partitioned tables can contain large amounts of data and simplify the rolling in and rolling out of table data.

Temporal tables

These types of tables are used to associate time-based state information to your data. Data in tables that do not use temporal support represents the present, while data in temporal tables is valid for a period defined by the database system, customer applications, or both. For example, a database can store the history of a table (deleted rows or the original values of rows that have been updated) so you can query the past state of your data. You can also assign a date range to a row of data to indicate when it is deemed to be valid by your application or business rules.

Temporary tables

These types of tables are used as temporary work tables for various database operations. *Declared temporary tables* (DGTTs) do not appear in the system catalog, which makes them not persistent for use by, and not able to be shared with other applications. When the application using this table terminates or disconnects from the database, any data in the table is deleted and the table is dropped. By contrast, *created temporary tables* (CGTTs) do appear in the system catalog and are not required to be defined in every session where they are used. As a result, they are persistent and able to be shared with other applications across different connections.

Neither type of temporary table supports

- User-defined reference or user-defined structured type columns
- LONG VARCHAR columns

In addition XML columns cannot be used in created temporary tables.

Materialized query tables

These types of tables are defined by a query that is also used to determine the data in the table. Materialized query tables can be used to improve the performance of queries. If the database manager determines that a portion of a query can be resolved using a summary table, the database manager can rewrite the query to use the summary table. This decision is based on database configuration settings, such as the CURRENT REFRESH AGE and the CURRENT QUERY OPTIMIZATION special registers. A summary table is a specialized type of materialized query table.

You can create all of the preceding types of tables using the CREATE TABLE statement.

Depending on what your data is going to look like, you might find one table type offers specific capabilities that can optimize storage and query performance. For example, if you have data records that are loosely clustered (not monotonically increasing), consider using a regular table and indexes. If you have data records that have duplicate (but not unique) values in the key, do not use a range-clustered table. Also, if you cannot afford to preallocate a fixed amount of storage on disk for the range-clustered tables you might want, do not use this type of table. If you have data that has the potential for being clustered along multiple dimensions, such as a table tracking retail sales by geographic region, division and supplier, a multidimensional clustering table might suit your purposes.

In addition to the various table types described previously, you also have options for such characteristics as *partitioning*, which can improve performance for tasks such as rolling in table data. Partitioned tables can also hold much more information than a regular, nonpartitioned table. You can also use capabilities such as *compression*, which can help you significantly reduce your data storage costs.

Designing tables

When designing tables, you must be familiar with certain concepts, determine the space requirements for tables and user data, and determine whether you will take advantage of certain features, such as compression and optimistic locking.

When designing partitioned tables, you must be familiar with the partitioning concepts, such as:

- Data organization schemes
- table-partitioning keys
- Keys used for distributing data across data partitions
- Keys used for MDC dimensions

For these and other partitioning concepts, see “Table partitioning and data organization schemes” on page 232.

Data types and table columns

When you create your table, you must indicate what type of data each column will store. By thinking carefully about the nature of the data you are going to be managing, you can set your tables up in a way that will give you optimal query performance, minimize physical storage requirements, and provide you with specialized capabilities for manipulating different kinds of data, such as arithmetic operations for numeric data, or comparing date or time values to one another.

Figure 18 on page 204 shows the data types that are supported by DB2 databases.

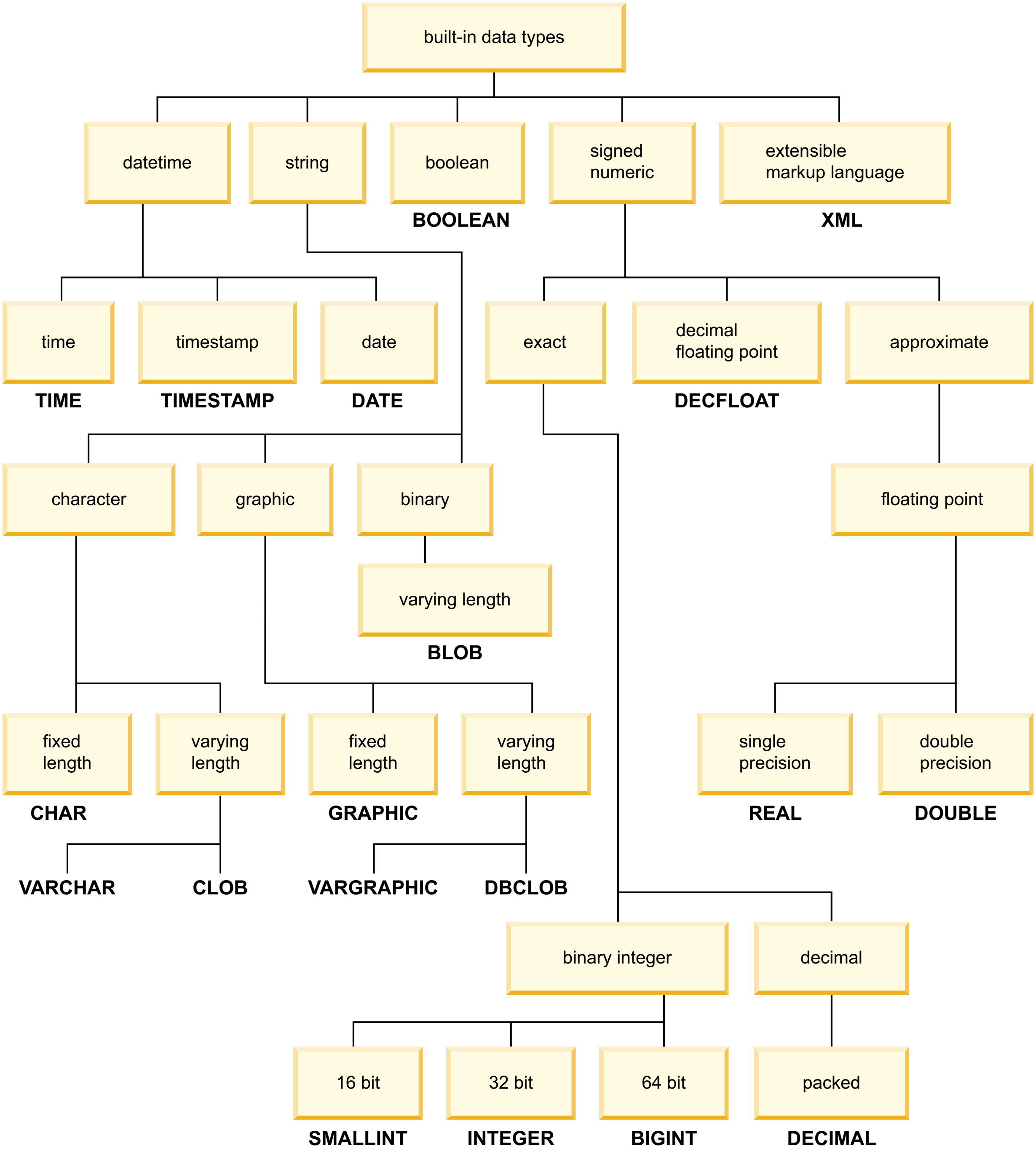


Figure 18. Built-in data types

When you declare your database columns all of these data tyoes are available for you to choose from. In addition to the built-in types, you can also create your own *user-defined* data types that are based on the built-in types. For example, if you might choose to represent an employee with name, job title, job level, hire date and salary attributes with a user-defined *structured* type that incorporates VARCHAR (name, job title), SMALLINT (job level), DATE (hire date) and DECIMAL (salary) data.

Generated columns

A generated column is defined in a table where the stored value is computed using an expression, rather than being specified through an insert or update operation.

When creating a table where it is known that certain expressions or predicates will be used all the time, you can add one or more generated columns to that table. By using a generated column there is opportunity for performance improvements when querying the table data.

For example, there are two ways in which the evaluation of expressions can be costly when performance is important:

1. The evaluation of the expression must be done many times during a query.
2. The computation is complex.

To improve the performance of the query, you can define an additional column that would contain the results of the expression. Then, when issuing a query that includes the same expression, the generated column can be used directly; or, the query rewrite component of the optimizer can replace the expression with the generated column.

Where queries involve the joining of data from two or more tables, the addition of a generated column can allow the optimizer a choice of possibly better join strategies.

Generated columns will be used to improve performance of queries. As a result, generated columns will likely be added after the table has been created and populated.

Examples

The following is an example of defining a generated column on the CREATE TABLE statement:

```
CREATE TABLE t1 (c1 INT,
                 c2 DOUBLE,
                 c3 DOUBLE GENERATED ALWAYS AS (c1 + c2)
                 c4 GENERATED ALWAYS AS
                    (CASE WHEN c1 > c2 THEN 1 ELSE NULL END))
```

After creating this table, indexes can be created using the generated columns. For example,

```
CREATE INDEX i1 ON t1(c4)
```

Queries can take advantage of the generated columns. For example,

```
SELECT COUNT(*) FROM t1 WHERE c1 > c2
```

can be written as:

```
 SELECT COUNT(*) FROM t1 WHERE c4 IS NOT NULL
```

Another example:

```
SELECT c1 + c2 FROM t1 WHERE (c1 + c2) * c1 > 100
```

can be written as:

```
SELECT c3 FROM t1 WHERE c3 * c1 > 100
```

Hidden columns

When a table column is defined with the implicitly hidden attribute, that column is unavailable unless it is explicitly referenced. For example, if a SELECT * query is run against a table, implicitly hidden columns are not returned in the result table. An implicitly hidden column can always be referenced explicitly wherever a column name can be specified.

In cases where columns and their entries are generated by the database manager, defining such columns as IMPLICITLY HIDDEN can minimize any potential negative impact on your applications. For example, a system-period temporal table has three columns whose values are generated by the database manager. The database manager uses these columns to preserve historical versions of each table row. Most business applications would work with the historical data, but would rarely work with these three generated columns. Hiding these columns from your applications could reduce application processing time.

When inserting data into a table, an INSERT statement without a column list does not expect values for any implicitly hidden columns. In such cases, if the input includes a value for an implicitly hidden column, that value does not have a corresponding target column and an error is returned (SQLSTATE 42802). Because an INSERT statement without a column list does not include values for implicitly hidden columns, any columns that are defined as implicitly hidden and NOT NULL must have a defined default value

When populating a table with data from an input file, utilities like IMPORT, INGEST, and LOAD require that you specify whether data for the hidden columns is included in the operation. If a column list is not specified, data movement utilities must use the *implicitlyhiddeninclude* or *implicitlyhiddenmissing* file type modifiers when working with tables that contain implicitly hidden columns. You can also use the DB2_DMU_DEFAULT registry variable to set the default behavior when data movement utilities encounter tables with implicitly hidden columns. Similarly, EXPORT requires that you specify whether data for the hidden columns is included in the operation.

The implicitly hidden attribute can be defined on a table column using the CREATE TABLE statement for new tables, or the ALTER TABLE statement for existing tables. If a table is created using a CREATE TABLE statement with the LIKE clause, any implicitly hidden columns in the source table are inherited by the new table. The ALTER TABLE statement can be used to change hidden columns to not hidden or to change not hidden columns to hidden. Altering a table to change the hidden attribute of some columns can impact the behavior of data movement utilities that are working with the table. For example, this might mean that a load operation that ran successfully before the table was altered to define some hidden columns, now returns an error (SQLCODE -2437).

The list of names identifying the columns of a result table from a SELECT query run with the *exposed-name.** option does not include any implicitly hidden columns. A SELECT query run with the `order-by-clause` can include implicitly hidden columns in the *simple-column-name*.

If an implicitly hidden column is explicitly referenced in a materialized query table definition, that column will be a part of the materialized query table. However the column in the materialized query table does not inherit the implicitly hidden attribute. This same behaviour applies to views and tables created with the as-result-table clause.

An implicitly hidden column can be explicitly referenced in a CREATE INDEX statement, ALTER TABLE statement, or in a referential constraint.

A transition variable exists for any column defined as implicitly hidden. In the body of a trigger, a transition variable that corresponds to an implicitly hidden column can be referenced.

Implicitly hidden columns are not supported in created temporary tables and declared temporary tables.

Hidden columns for a table can be displayed using the DESCRIBE command.

```
DESCRIBE TABLE tablename SHOW DETAIL
```

Example

- *Example 1:* In the following statement, a table is created with an implicitly hidden column.

  ```
  CREATE TABLE CUSTOMER
  (
   CUSTOMERNO       INTEGER NOT NULL,
   CUSTOMERNAME     VARCHAR(80),
   PHONENO          CHAR(8) IMPLICITLY HIDDEN
  );
  ```

  A SELECT * only returns the column entries for CUSTOMERNO and CUSTOMERNAME. For example:

  ```
  A123, ACME
  B567, First Choice
  C345, National Chain
  ```

  Entries for the PHONENO column are hidden unless explicitly referenced.

  ```
  SELECT CUSTOMERNO, CUSTOMERNAME, PHONENO
     FROM CUSTOMER
  ```

- *Example 2:* If the database table contains implicitly hidden columns, you must specify whether data for the hidden columns is included in data movement operations. The following example uses LOAD to show the different methods to indicate if data for hidden columns is included:
  - Use *insert-column* to explicitly specify the columns into which data is to be inserted.

    ```
    db2 load from delfile1 of del
       insert into table1 (c1, c2, c3,...)
    ```

  - Use one of the hidden column file type modifiers: specify **implicitlyhiddeninclude** when the input file contains data for the hidden columns, or **implicitlyhiddenmissing** when the input file does not.

    ```
    db2 load from delfile1 of del modified by implicitlyhiddeninclude
       insert into table1
    ```

  - Use the DB2_DMU_DEFAULT registry variable on the server-side to set the behavior when data movement utilities encounter tables with implicitly hidden columns.

    ```
    db2set DB2_DMU_DEFAULT=IMPLICITLYHIDDENINCLUDE
    db2 load from delfile1 of del insert into table1
    ```

Auto numbering and identifier columns

An identity column provides a way for DB2 to automatically generate a unique numeric value for each row that is added to the table.

When creating a table in which you must uniquely identify each row that will be added to the table, you can add an identity column to the table. To guarantee a unique numeric value for each row that is added to a table, you should define a unique index on the identity column or declare it a primary key.

Other uses of an identity column are an order number, an employee number, a stock number, or an incident number. The values for an identity column can be generated by the DB2 database manager: ALWAYS or BY DEFAULT.

An identity column defined as GENERATED ALWAYS is given values that are always generated by the DB2 database manager. Applications are not allowed to provide an explicit value. An identity column defined as GENERATED BY DEFAULT gives applications a way to explicitly provide a value for the identity column. If the application does not provide a value, then DB2 will generate one. Since the application controls the value, DB2 cannot guarantee the uniqueness of the value. The GENERATED BY DEFAULT clause is meant for use for data propagation where the intent is to copy the contents of an existing table; or, for the unload and reloading of a table.

Once created, you first have to add the column with the DEFAULT option to get the existing default value. Then you can ALTER the default to become an identity column.

If rows are inserted into a table with explicit identity column values specified, the next internally generated value is not updated, and might conflict with existing values in the table. Duplicate values will generate an error message if the uniqueness of the values in the identity column is being enforced by a primary-key or a unique index that has been defined on the identity column.

To define an identity column on a new table, use the AS IDENTITY clause on the CREATE TABLE statement.

Example

The following is an example of defining an identity column on the CREATE TABLE statement:

```
CREATE TABLE table (col1 INT,
                    col2 DOUBLE,
                    col3 INT NOT NULL GENERATED ALWAYS AS IDENTITY
                                      (START WITH 100, INCREMENT BY 5))
```

In this example the third column is the identity column. You can also specify the value used in the column to uniquely identify each row when added. Here the first row entered has the value of "100" placed in the column; every subsequent row added to the table has the associated value increased by five.

Constraining column data with constraints, defaults, and null settings

Data often must adhere to certain restrictions or rules. Such restrictions might apply to single pieces of information, such as the format and sequence numbers, or they might apply to several pieces of information.

About this task

Nullability of column data values

Null values represent unknown states. By default, all of the built-in data types support the presence of null values. However, some business rules might dictate that a value must always be provided for some columns, for example, emergency information. For this condition, you can use the NOT NULL constraint to ensure that a given column of a table is never assigned the null value. Once a NOT NULL constraint has been defined for a particular column, any insert or update operation that attempts to place a null value in that column will fail.

Default column data values

Just as some business rules dictate that a value must always be provided, other business rules can dictate what that value should be, for example, the gender of an employee must be either M or F. The column default constraint is used to ensure that a given column of a table is always assigned a predefined value whenever a row that does not have a specific value for that column is added to the table. The default value provided for a column can be null, a constraint value that is compatible with the data type of the column, or a value that is provided by the database manager. For more information, see: "Default column and data type definitions."

Keys A key is a single column or a set of columns in a table or index that can be used to identify or access a specific row of data. Any column can be part of a key and the same column can be part of more than one key. A key that consists of a single column is called an atomic key; a key that is composed of more than one column is called a composite key. In addition to having atomic or composite attributes, keys are classified according to how they are used to implement constraints:

- A unique key is used to implement unique constraints.
- A primary key is used to implement entity integrity constraints. (A primary key is a special type of unique key that does not support null values.)
- A foreign key is used to implement referential integrity constraints. (Foreign keys must reference primary keys or unique keys; foreign keys do not have corresponding indexes.)

Keys are normally specified during the declaration of a table, an index, or a referential constraint definition.

Constraints

Constraints are rules that limit the values that can be inserted, deleted, or updated in a table. There are check constraints, primary key constraints, referential constraints, unique constraints, unique key constraints, foreign key constraints, and informational constraints. For details about each of these types of constraints, see: Chapter 21, "Constraints," on page 299 or "Types of constraints" on page 299.

Default column and data type definitions

Certain columns and data types have predefined or assigned default values.

For example, default column values for the various data types are as follows:

- *NULL*
- *0* Used for small integer, integer, decimal, single-precision floating point, double-precision floating point, and decimal floating point data type.
- *Blank:* Used for fixed-length and fixed-length double-byte character strings.
- *Zero-length string:* Used for varying-length character strings, binary large objects, character large objects, and double-byte character large objects.

- *Date:* This the system date at the time the row is inserted (obtained from the CURRENT_DATE special register). When a date column is added to an existing table, existing rows are assigned the date January, 01, 0001.
- *Time or Timestamp:* This is the system time or system date/time of the at the time the statement is inserted (obtained from the CURRENT_TIME special register). When a time column is added to an existing table, existing rows are assigned the time 00:00:00 or a timestamp that contains the date January, 01, 0001 and the time 00:00:00.

  Note: All the rows get the same default time/timestamp value for a given statement.
- *Distinct user-defined data type:* This is the built-in default value for the base data type of the distinct user-defined data type (cast to the distinct user-defined data type.

Ordering columns to minimize update logging

When you define columns using the CREATE TABLE statement, consider the order of the columns, particularly for update-intensive workloads. Columns which are updated frequently should be grouped together, and defined toward or at the end of the table definition. This results in better performance, fewer bytes logged, and fewer log pages written, as well as a smaller active log space requirement for transactions performing a large number of updates.

The database manager does not automatically assume that columns specified in the SET clause of an UPDATE statement are changing in value. In order to limit index maintenance and the amount of the row which needs to be logged, the database compares the new column value against the old column value to determine if the column is changing. Only the columns that are changing in value are treated as being updated. Exceptions to this UPDATE behavior occur for columns where the data is stored outside of the data row (long, LOB, ADT, and XML column types), or for fixed-length columns when the registry variable DB2ASSUMEUPDATE is enabled. For these exceptions, the column value is assumed to be changing so no comparison will be made between the new and old column value.

There are four different types of UPDATE log records.

- Full before and after row image logging. The entire before and after image of the row is logged. This is the only type of logging performed on tables enabled with DATA CAPTURE CHANGES, and results in the most number of bytes being logged for an update to a row.
- Full before row image, changed bytes, and for size increasing updates the new data appended to end of the row. This is logged for databases supporting Currently Committed when DATA CAPTURE CHANGES is not in effect for the table, when update is the first action against this row for a transaction. This logs the before image required for Currently Committed and the minimum required on top of that for redo/undo. Ordering frequently updated columns at the end minimizes the logging for the changed portion of the row.
- Full XOR logging. The XOR differences between the before and after row images, from the first byte that is changing until the end of the smaller row, then any residual bytes in the longer row. This results in less logged bytes than the full before and after image logging, with the number of bytes of data beyond the log record header information being the size of the largest row image.
- Partial XOR logging. The XOR differences between the before and after row images, from the first byte that is changing until the last byte that is changing.

Byte positions can be first or last bytes of a column. This results in the least number of bytes being logged and the most efficient type of log record for an update to a row.

For the first two types of UPDATE log records listed previously, when DATA CAPTURE CHANGES is not enabled on the table, the amount of data that is logged for an update depends on:

- The proximity of the updated columns (COLNO)
- Whether the updated columns are fixed in length or variable length
- Whether row compression (COMPRESS YES) is enabled

When the total length of the row is not changing, even when row compression is enabled, the database manager computes and writes the optimal partial XOR log record.

When the total length of the row is changing, which is common when variable-length columns are updated and row compression is enabled, the database manager determines which byte is first to be changed and write a full XOR log record.

Space requirements for tables

When designing tables, you need to take into account the space requirements for the data the tables will contain. In particular, you must pay attention to columns with larger data types, such as LOB or XML.

Large object (LOB) data

Large object (LOB) data is stored in two separate table objects that are structured differently than the storage space for other data types. To estimate the space required by LOB data, you must consider the two table objects used to store data defined with these data types:

- *LOB Data Objects:* Data is stored in 64 MB areas that are broken up into segments whose sizes are "powers of two" times 1024 bytes. (Hence these segments can be 1024 bytes, 2048 bytes, 4096 bytes, and so on, up to 64 MB.)

  To reduce the amount of disk space used by LOB data, you can specify the COMPACT option on the lob-options clause of the CREATE TABLE and the ALTER TABLE statements. The COMPACT option minimizes the amount of disk space required by allowing the LOB data to be split into smaller segments. This process does not involve data compression, but simply uses the minimum amount of space, to the nearest 1 KB boundary. Using the COMPACT option can result in reduced performance when appending to LOB values.

  The amount of free space contained in LOB data objects is influenced by the amount of update and delete activity, as well as the size of the LOB values being inserted.
- *LOB Allocation Objects:* Allocation and free space information is stored in allocation pages that are separated from the actual data. The number of these pages is dependent on the amount of data, including unused space, allocated for the large object data. The extra space is calculated as follows:

Table 19. Allocation page extra space based on the page size

Page size	Allocation pages
4 KB	One page for every 4 MB, plus one page for every 1 GB

Table 19. Allocation page extra space based on the page size (continued)

Page size	Allocation pages
8 KB	One page for every 8 MB, plus one page for every 2 GB
16 KB	One page for every 16 MB, plus one page for every 4 GB
32 KB	One page for every 32 MB, plus one page for every 8 GB

If character data is less than the page size, and it fits into the record along with the rest of the data, the CHAR, GRAPHIC, VARCHAR, or VARGRAPHIC data types should be used instead of BLOB, CLOB, or DBCLOB.

Note: Some LOB data can be placed into the base table row through the use of the INLINE LENGTH option of the CREATE and ALTER TABLE statements.

Long field (LF) data

Long field (LF) data is stored in a separate table object that is structured differently than the storage space for other data types. Data is stored in 32-KB areas that are broken up into segments whose sizes are "powers of two" times 512 bytes. (Hence these segments can be 512 bytes, 1024 bytes, 2048 bytes, and so on, up to 32 768 bytes.)

Long field data types (LONG VARCHAR or LONG VARGRAPHIC) are stored in a way that enables free space to be reclaimed easily. Allocation and free space information is stored in 4 KB allocation pages, which appear infrequently throughout the object.

The amount of unused space in the object depends on the size of the long field data, and whether this size is relatively constant across all occurrences of the data. For data entries larger than 255 bytes, this unused space can be up to 50 percent of the size of the long field data.

If character data is less than the page size, and it fits into the record along with the rest of the data, the CHAR, GRAPHIC, VARCHAR, or VARGRAPHIC data types should be used instead of LONG VARCHAR or LONG VARGRAPHIC.

System catalog tables

System catalog tables are created when a database is created. The system tables grow as database objects and privileges are added to the database. Initially, they use approximately 3.5 MB of disk space.

The amount of space allocated for the catalog tables depends on the type of table space, and the extent size of the table space containing the catalog tables. For example, if a DMS table space with an extent size of 32 is used, the catalog table space is initially allocated 20 MB of space. Note: For databases with multiple partitions, the catalog tables reside only on the database partition from which the `CREATE DATABASE` command was issued. Disk space for the catalog tables is only required for that database partition.

Temporary tables

Some statements require temporary tables for processing (such as a work file for sorting operations that cannot be done in memory). These temporary tables require disk space; the amount of space required is dependent upon the size, number, and nature of the queries, and the size of returned tables.

Your work environment is unique which makes the determination of your space requirements for temporary tables difficult to estimate. For example, more space can appear to be allocated for system temporary table spaces than is actually in use due to the longer life of various system temporary tables. This could occur when **DB2_SMS_TRUNC_TMPTABLE_THRESH** registry variable is used.

You can use the database system monitor and the table space query APIs to track the amount of work space being used during the normal course of operations.

You can use the **DB2_OPT_MAX_TEMP_SIZE** registry variable to limit the amount of temporary table space used by queries.

XML data

XML documents you insert into columns of type XML can reside either in the default storage object, or directly in the base table row. Base table row storage is under your control and is available only for small documents; larger documents are always stored in the default storage object.

Table page sizes

Rows of table data are organized into blocks that are called pages. Pages can be four sizes: 4, 8, 16, and 32 KB. Table data pages do not contain the data for columns that are defined with LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DCLOB, or XML data types. An exception is if the LOB or XML document is inlined by using INLINE LENGTH option of the column. The rows in a table data page do, however, contain a descriptor of these columns.

Note: Some LOB and XML data can be placed into the base table row by using the INLINE LENGTH option of the CREATE and ALTER TABLE statements.

All tables that are created within a table space of a particular size have a matching page size. A single table or index object can be as large as 64 TB, assuming a 32-KB page size.

Larger page sizes can reduce the number of levels in the index. Larger pages support rows of greater length. Using the default of 4-KB pages, tables are restricted to 500 columns. Larger page sizes (8 KB, 16 KB, and 32 KB) support 1012 columns. The maximum size of the table space is proportional to the page size of the table space.

Page size defines the size of pages that are used for the table space. The page size limits the row length and column count of tables according to the figures shown in Table 1.

Table 20. Implications of page size with regular table space

Page size	Row count limit	Row size limit	Column count limit	
4 KB	255	4 005 bytes	500	
8 KB	255	8 101 bytes	1 012	
16 KB	255	16 293 bytes	1 012	
32 KB	255	32 677 bytes	1 012	

Note: A large table space can support more than 255 rows per data page. A table that is created in a large table space can be larger than a table created in a regular table space. A large table space stores all permanent data just as a regular table space does. The result is a better use of space on data pages.

To determine the page size for a table space, consider the following points:

- For OLTP applications that run random row read and write operations, a smaller page size is preferable. It takes less buffer pool space with unwanted rows.
- For DSS applications that access large numbers of consecutive rows at a time, a larger page size is better. A larger page size reduces the number of I/O requests that are required to read a specific number of rows. There is, however, an exception to this rule. If your row size is smaller than `pagesize / maximum rows`, there is used space on each page. In this situation, a smaller page size might be more appropriate.

Space requirements for user table data

By default, table data is stored based on the table space page size in which the table is in. Each page (regardless of page size) contains 68 bytes of overhead for the database manager. A row will *not* span multiple pages. You can have a maximum of 500 columns when using a 4-KB page size.

Table data pages *do not* contain the data for columns defined with LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, or XML data types. The rows in a table data page do, however, contain a descriptor for these columns.

Note: Some LOB data can be placed into the base table row through the use of the INLINE LENGTH option of the CREATE and ALTER TABLE statements.

Rows are usually inserted into a regular table in first-fit order. The file is searched (using a free space map) for the first available space that is large enough to hold the new row. When a row is updated, it is updated in place, unless there is insufficient space left on the page to contain it. If this is the case, a record is created in the original row location that points to the new location in the table file of the updated row.

If the ALTER TABLE statement is issued with the `APPEND ON` option, data is always appended, and information about any free space on the data pages is not kept.

If the table has a clustering index defined on it, the database manager will attempt to physically cluster the data according to the key order of that clustering index. When a row is inserted into the table, the database manager will first look up its key value in the clustering index. If the key value is found, the database manager attempts to insert the record on the data page pointed to by that key; if the key value is not found, the next higher key value is used, so that the record is inserted on the page containing records having the next higher key value. If there is insufficient space on the target page in the table, the free space map is used to search neighboring pages for space. Over time, as space on the data pages is completely used up, records are placed further and further from the target page in the table. The table data would then be considered unclustered, and a table reorganization can be used to restore clustered order.

If the table is a multidimensional clustering (MDC) table, the database manager will guarantee that records are always physically clustered along one or more defined dimensions, or clustering indexes. When an MDC table is defined with certain dimensions, a block index is created for each of the dimensions, and a composite block index is created which maps cells (unique combinations of dimension values) to blocks. This composite block index is used to determine to which cell a particular record belongs, and exactly which blocks or extents in the table contains records belonging to that cell. As a result, when inserting records, the database manager searches the composite block index for the list of blocks containing records having the same dimension values, and limits the search for space to those blocks only. If the cell does not yet exist, or if there is insufficient space in the cell's existing blocks, then another block is assigned to the cell and the record is inserted into it. A free space map is still used within blocks to quickly find available space in the blocks.

The number of 4-KB pages for each user table in the database can be estimated by calculating:

```
ROUND DOWN(4028/(average row size + 10)) = records_per_page
```

and then inserting the result into:

```
(number_of_records/records_per_page) * 1.1 = number_of_pages
```

where the average row size is the sum of the average column sizes, and the factor of "1.1" is for overhead.

Note: This formula provides only an estimate. The estimate's accuracy is reduced if the record length varies because of fragmentation and overflow records.

You also have the option to create buffer pools or table spaces that have an 8 KB, 16 KB, or 32 KB page size. All tables created within a table space of a particular size have a matching page size. A single table or index object can be as large as 64 TB, assuming a 32 KB page size. You can have a maximum of 1012 columns when using an 8 KB, 16 KB, or 32 KB page size. The maximum number of columns is 500 for a 4-KB page size. Maximum row lengths also vary, depending on page size:

- When the page size is 4-KB, the row length can be up to 4005 bytes.
- When the page size is 8 KB, the row length can be up to 8101 bytes.
- When the page size is 16 KB, the row length can be up to 16 293 bytes.
- When the page size is 32 KB, the row length can be up to 32 677 bytes.

A larger page size facilitates a reduction in the number of levels in any index. If you are working with OLTP (online transaction processing) applications, that perform random row reads and writes, a smaller page size is better, because it consumes less buffer space with undesired rows. If you are working with DSS (decision support system) applications, which access large numbers of consecutive rows at a time, a larger page size is better because it reduces the number of I/O requests required to read a specific number of rows.

You cannot restore a backup image to a different page size.

You cannot import IXF data files that represent more than 755 columns.

Declared or created temporary tables can be declared or created only in their own user temporary table space type. There is no default user temporary table space. The temporary tables are dropped implicitly when an application disconnects from the database, and estimates of the space requirements for these tables should take this into account.

Storing LOBs inline in table rows

Large objects (LOBs) are generally stored in a location separate from the table row that references them. However, you can choose to include a LOB to 32 673 bytes long inline in a base table row to simplify access to it.

It can be impractical (and depending on the data, impossible) to include large data objects in base table rows. Figure 19 shows an example of an attempt to include LOBs within a row, and why doing so can be a problem. In this example, the row is defined as having two LOB columns, 500 and 145 kilobytes in length. However, the maximum row size for a DB2 table is 32 kilobytes; so such a row definition could never, in fact, be implemented.

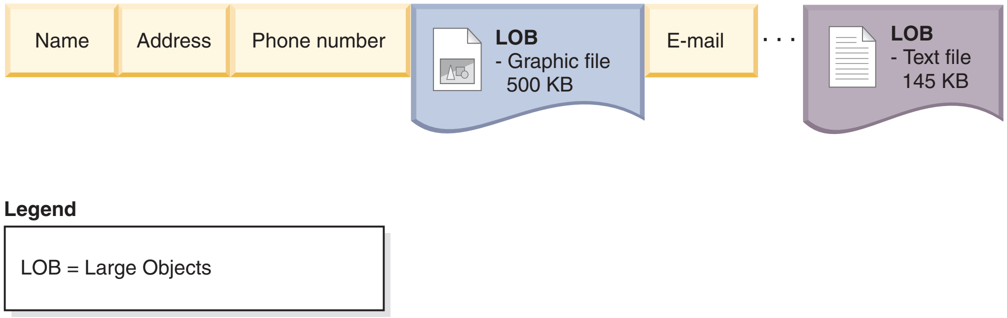


Figure 19. The problem of including LOB data within base table rows

To reduce the difficulties associated with working with LOBs, they are treated differently from other data types. Figure 20 on page 217, shows that only a LOB descriptor is placed in the base table row, rather than the LOB itself. Each of the LOBs themselves are stored in a separate LOBs location controlled by the database manager. In this arrangement, the movement of rows between the buffer pool and disk storage will take less time for rows with LOB descriptors than they would if they included the complete LOBs.

However, manipulation of the LOB data then becomes more difficult because the actual LOB is stored in a location separate from the base table rows.

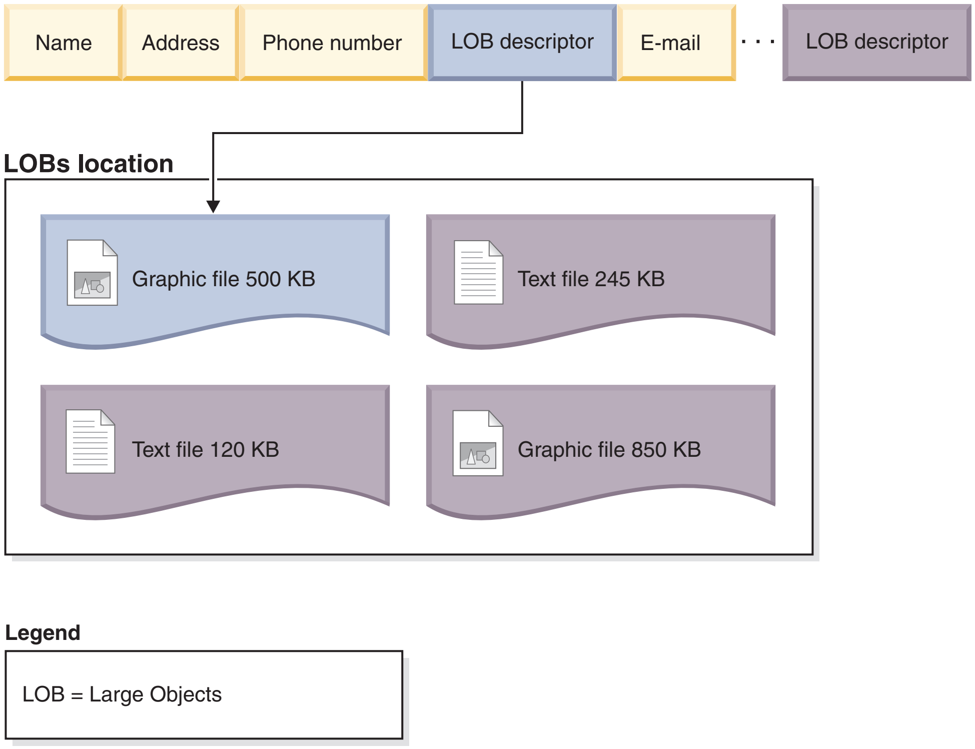


Figure 20. LOB descriptors within the base table row refer to the LOBs within the separate LOBs location

To simplify the manipulation of smaller LOBs, you can choose to have LOB data that falls below a size threshold that you specify included inline within the base table rows. These LOB data types can then be manipulated as part of the base table row, which makes operations such as movement to and from the buffer pool simpler. In addition, the inline LOBs would qualify for row compression if row compression was enabled.

The INLINE LENGTH option of the CREATE and ALTER TABLE statements allows LOB data smaller than a length restriction that you specify to be included in the base table row. By default, even if you don't specify an explicit value for INLINE LENGTH, LOBs smaller than the maximum size LOB descriptor for the column are always included in the base table row.

With inline LOBs then, you can have base table rows as shown in Figure 21 on page 218.

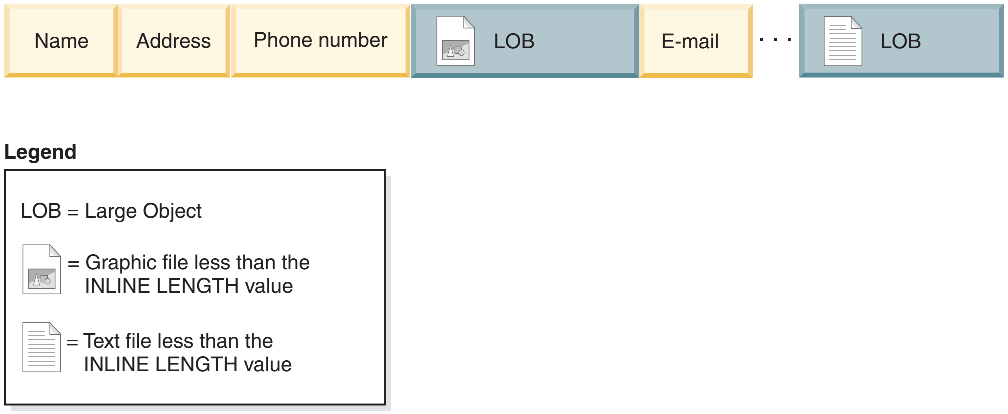

Name
Address
Phone number
LOB
E-mail
. . .
LOB
Legend
LOB = Large Object
= Graphic file less than the INLINE LENGTH value
= Text file less than the INLINE LENGTH value

Figure 21. Small LOBs included within base table rows

When you are considering the threshold to choose for including LOBs inline, take into account the current pagesize for your database, and whether inline LOBs will cause the row size to exceed the current page size. The maximum size for a row in a table is 32 677 bytes. However, each inline LOB has 4 bytes of extra storage required. So each LOB you store inline reduces the available storage in the row by 4 bytes. Thus the maximum size for an inline LOB is 32 673 bytes.

Note: In the same way that LOBs can be stored inline, it's also possible to store XML data inline as well.

Table compression

You can use less disk space for your tables by taking advantage of the DB2 table compression capabilities. Compression saves disk storage space by using fewer database pages to store data.

Also, because you can store more rows per page, fewer pages must be read to access the same amount of data. Therefore, queries on a compressed table need fewer I/O operations to access the same amount of data. Since there are more rows of data on a buffer pool page, the likelihood that needed rows are in the buffer pool increases. For this reason, compression can improve performance through improved buffer pool hit ratios. In a similar way, compression can also speed up backup and restore operations, as fewer pages of need to be transferred to the backup or restore the same amount of data.

You can use compression with both new and existing tables. Temporary tables are also compressed automatically, if the database manager deems it to be advantageous to do so.

There are two main types of data compression availble for tables:

- Row compression (available with a license for the DB2 Storage Optimization Feature).
- Value compression

For a particular table, you can use row compression and value compression together or individually. However, you can use only one type of row compression for a particular table.

Value compression

Value compression optimizes space usage for the representation of data, and the storage structures used internally by the database management system to store data. Value compression involves removing duplicate entries for a value, and only storing one copy. The stored copy keeps track of the location of any references to the stored value.

When creating a table, you can use the optional VALUE COMPRESSION clause of the CREATE TABLE statement to specify that the table is to use value compression. You can enable value compression in an existing table with the ACTIVATE VALUE COMPRESSION clause of the ALTER TABLE statement. To disable value compression in a table, you use the DEACTIVATE VALUE COMPRESSION clause of the ALTER TABLE statement.

When VALUE COMPRESSION is used, NULLs and zero-length data that has been assigned to defined variable-length data types (VARCHAR, VARGRAPHICS, LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, and DBCLOB) will not be stored on disk.

If VALUE COMPRESSION is used then the optional COMPRESS SYSTEM DEFAULT option can also be used to further reduce disk space usage. Minimal disk space is used if the inserted or updated value is equal to the system default value for the data type of the column, as the default value will not be stored on disk. Data types that support COMPRESS SYSTEM DEFAULT include all numeric type columns, fixed-length character, and fixed-length graphic string data types. This means that zeros and blanks can be compressed.

When using value compression, the byte count of a compressed column in a row might be larger than that of the uncompressed version of the same column. If your row size approaches the maximum allowed for your page size, you must ensure that sum of the byte counts for compressed and uncompressed columns does not exceed allowable row length of the table in the table space. For example, in a table space with 4 KB page size, the allowable row length is 4005 bytes. If the allowable row length is exceeded, the error message SQL0670N is returned. The formula used to determine the byte counts for compressed and uncompressed columns is documented as part of the CREATE TABLE statement.

If you deactivate value compression:

- COMPRESS SYSTEM DEFAULTS will also be deactivated implicitly, if it had previously been enabled
- The uncompressed columns might cause the row size to exceed the maximum allowed by the current page size of the current table space. If this occurs, the error messasge SQL0670N will be returned.
- Existing compressed data will remain compressed until the row is updated or you perform a table reorganization with the REORG command.

Row compression

Row compression uses a dictionary-based compression algorithm to replace recurring strings with shorter symbols within data rows.

There are two types of row compression that you can choose from:

- “Classic” row compression.
- Adaptive compression

Row compression is available with a license for the DB2 Storage Optimization Feature. Depending on the DB2 product edition that you have, this feature might be included, or it might be an option that you order separately.

Classic row compression

Classic row compression, sometimes referred to as *static compression* , compresses data rows by replacing patterns of values that repeat across rows with shorter symbol strings.

The benefits of using classic row compression are similar to those of adaptive compression, in that you can store data in less space, which can significantly save storage costs. Unlike adaptive compression, however, classic row compression uses only a table-level dictionary to store globally recurring patterns; it doesn't use the page-level dictionaries that are used to compress data dynamically.

How classic row compression works

Classic row compression uses a table-level compression dictionary to compress data by row. The dictionary is used to map repeated byte patterns from table rows to much smaller symbols; these symbols then replace the longer byte patterns in the table rows. The compression dictionary is stored with the table data rows in the data object portions of the table.

What data gets compressed?

Data that is stored in base table rows and log records is eligible for classic row compression. In addition, the data in XML storage objects is eligible for compression. You can compress LOB data that you place inline in a table row; however, storage objects for long data objects are not compressed.

Restriction: You cannot compress data in XML columns that you created with DB2 Version 9.5 or DB2 Version 9.1. However, you can compress inline XML columns that you add to a table using DB2 Version 9.7 or later, provided the table was created without XML columns in an earlier release of the product. If a table that you created in an earlier release already has one or more XML columns and you want to add a compressed XML column by using DB2 Version 9.7 or later, you must use the ADMIN_MOVE_TABLE stored procedure to migrate the table before you can use compression.

Turning classic row compression on or off

To use classic row compression, you must have a license for the DB2 Storage Optimization Feature. You compress table data by setting the COMPRESS attribute of the table to YES STATIC. You can set this attribute when you create the table by specifying the COMPRESS YES STATIC option for the CREATE TABLE statement. You can also alter an existing table to use compression by using the same option for the ALTER TABLE statement. After you enable compression, operations that add data to the table, such as an **INSERT**, **LOAD INSERT**, or **IMPORT INSERT** command operation, can use classic row compression. In addition, index compression is enabled for the table. Indexes are created as compressed indexes unless you specify otherwise and if they are the types of indexes that can be compressed.

Important: When you enable classic row compression for a table, you enable it for the entire table, even if a table comprises more than one table partition.

To disable compression for a table, use the ALTER TABLE statement with the COMPRESS NO option; rows that you subsequently add are not compressed. To extract the entire table, you must perform a table reorganization with the **REORG TABLE** command.

If you have a license for theDB2 Storage Optimization Feature, compression for temporary tables is enabled automatically. You cannot enable or disable compression for temporary tables.

Effects of update activity on logs and compressed tables

Depending upon update activity and which columns are updated within a data row, log usage might increase. For information about how to minimize the effects of update activity on logs, see ""Ordering columns to minimize update logging" on page 210".

If a row increases in size, the new version of the row might not fit on the current data page. Rather, the new image of the row is stored on an overflow page. To minimize the creation of pointer-overflow records, increase the percentage of each page that is to be left as free space after a reorganization by using the ALTER TABLE statement with the PCTFREE option. For example, if you set the PCTFREE option to 5% before you enabled compression, you might change it to 10% when you enable compression. Increasing the percentage of each page to be left as free space is especially important for data that is heavily updated.

Classic row compression for temporary tables

Compression for temporary tables is enabled automatically with the DB2 Storage Optimization Feature. When executing queries, the DB2 optimizer considers the storage savings and the impact on query performance that compression of temporary tables offers to determine whether it is worthwhile to use compression. If it is worthwhile, compression is used automatically. The minimum size that a table must be before compression is used is larger for temporary tables than for regular tables.

You can use the explain facility or the **db2pd** tool to see whether the optimizer used compression for temporary tables.

Reclaiming space that was freed by compression

You can reclaim space that was freed by compressing data. For more information, see "Reclaimable storage" on page 136.

Adaptive compression

Adaptive compression improves upon the compression rates that can be achieved using classic row compression by itself. Adaptive compression incorporates classic row compression; however, it also works on a page-by-page basis to further compress data. Of the various data compression techniques in the DB2 product, adaptive compression offers the most dramatic possibilities for storage savings.

How adaptive compression works

Adaptive compression actually uses two compression approaches. The first employs the same table-level compression dictionary used in classic row compression to compress data based on repetition within a sampling of data from the table as a whole. The second approach uses a page-level dictionary-based compression algorithm to compress data based on data repetition within each page of data. The dictionaries map repeated byte patterns to much smaller symbols; these symbols then replace the longer byte patterns in the table. The table-level compression dictionary is stored within the table object for which it is created, and is used to compress data throughout the table. The page-level compression dictionary is stored with the data in the data page, and is used to compression only the data within that page. For more information about the role each of these dictionaries in compressing data, see "Compression dictionaries" on page 228.

Note: You can specify that a table be compressed with classic row compression only by using a table-level compression dictionary. However, you cannot specify that tables be compressed by using only page-level compression dictionaries. Adaptive compression uses both table-level and page-level compression dictionaries.

Data that is eligible for compression

Data that is stored *within* data rows, including inlined LOB or XML values, can be compressed with both adaptive and classic row compression. XML storage objects can be compressed using static compression. However storage objects for long data objects that are stored outside table rows is not compressed. In addition, though log records themselves are not compressed, the amount of log data written as a result of insert, update or delete operations is reduced by virtue of the rows being compressed.

Restriction: You cannot compress data in XML columns that you created with DB2 Version 9.5 or DB2 Version 9.1. However, you can compress inline XML columns that you add to a table using DB2 Version 9.7 or later, provided the table was created without XML columns in an earlier release of the product. If a table that you created in an earlier release already has one or more XML columns and you want to add a compressed XML column by using DB2 Version 9.7 or later, you must use the ADMIN_MOVE_TABLE stored procedure to migrate the table before you can use compression.

Turning adaptive compression on or off

To use adaptive compression, you must have a license for the DB2 Storage Optimization Feature. You compress table data by setting the COMPRESS attribute of the table to YES. You can set this attribute when you create the table by specifying the COMPRESS YES option for the CREATE TABLE statement. You can also alter an existing table to use compression by using the same option for the ALTER TABLE statement. After you enable compression, operations that add data to the table, such as an **INSERT**, **LOAD INSERT**, or **IMPORT INSERT** command operation, can use adaptive compression. In addition, index compression is enabled for the table. Indexes are created as compressed indexes unless you specify otherwise and if they are the types of indexes that can be compressed.

Important: When you enable adaptive compression for a table, you enable it for the entire table, even if the table comprises more than one table partition.

To disable compression for a table, use the ALTER TABLE statement with the COMPRESS NO option; rows that you later add are not compressed. Existing rows remain compressed. To extract the entire table after you turn off compression, you must perform a table reorganization with the **REORG TABLE** command.

If you apply the licence for the DB2 Storage Optimization Feature, compression for temporary tables is enabled automatically if the database manager deems it valuable. You cannot enable or disable compression for temporary tables.

Effects of update activity on logs and compressed tables

Depending upon update activity and the position of updates in a data row, log usage might increase. For information about the impact that the order of columns in a table has on update logging, see ""Ordering columns to minimize update logging" on page 210".

If a row increases in size after adding new data to it, the new version of the row might not fit on the current data page. Rather, the new image of the row is stored on an overflow page. To minimize the creation of pointer-overflow records, increase the percentage of each page that is to be left as free space after a reorganization by using the ALTER TABLE statement with the PCTFREE option. For example, if you set the PCTFREE option to 5% before you enabled compression, you might change it to 10% when you enable compression. Increasing the percentage of each page to be left as free space is especially important for data that is heavily updated.

Compression for temporary tables

Compression for temporary tables is enabled automatically with the DB2 Storage Optimization Feature. Only classic row compression is used for temporary tables.

System temporary tables
When executing queries, the DB2 optimizer considers the storage savings and the impact on query performance that compression of system-created temporary tables offers to determine whether it is worthwhile to use compression. If it is worthwhile, classic row compression is used automatically. The minimum size that a table must be before compression is used is larger for temporary tables than for regular tables.

User-created temporary tables
Created global temporary tables (CGTTs) and declared global temporary tables (DGTTs) are always compressed using classic row compression.

You can use the explain facility or the **db2pd** tool to see whether the optimizer used compression for system temporary tables.

Reclaiming space that was freed by compression

You can reclaim space that has been freed by compressing data. For more information, see "Reclaimable storage" on page 136.

Estimating storage savings offered by adaptive or classic row compression

You can view an estimate of the storage savings adaptive or classic row compression can provide for a table by using the ADMIN_GET_TAB_COMPRESS_INFO table function.

Before you begin

The estimated savings that adaptive or classic row compression offers depend on the statistics generated by running the **RUNSTATS** command. To get the most accurate estimate of the savings that can be achieved, run the **RUNSTATS** command before you perform the following steps.

Procedure

To estimate the storage savings adaptive or classic row compression can offer using the ADMIN_GET_TAB_COMPRESS_INFO table function:

1. Formulate a SELECT statement that uses the ADMIN_GET_TAB_COMPRESS_INFO table function. For example, for a table named SAMPLE1.T1, enter:

```
SELECT * FROM TABLE(SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO('SAMPLE1', 'T1'))
```

2. Execute the SELECT statement. Executing the statement shown in Step 1 might yield a report like the following:

```
TABSCHEMA TABNAME    DBPARTITIONNUM DATAPARTITIONID OBJECT_TYPE ROWCOMPMODE ...
--------- ---------- -------------- --------------- ----------- ----------- ...
SAMPLE1   T1         0              0               DATA        A           ...

  1 record(s) selected.
PCTPAGESSAVED_CURRENT AVGROWSIZE_CURRENT PCTPAGESSAVED_STATIC ...
--------------------- ------------------ -------------------- ...
                   96                 24                   81 ...
AVGROWSIZE_STATIC PCTPAGESSAVED_ADAPTIVE AVGROWSIZE_ADAPTIVE
----------------- ---------------------- -------------------
              148                     93                  44
```

Creating a table that uses compression

When you create a new table by issuing the CREATE TABLE statement, you have the option to compress the data contained in table rows.

Before you begin

You must decide which type of compression you want to use: adaptive compression, classic row compression, value compression, or a combination of value compression with either of the two types of row compression. Adaptive compression and classic row compression almost always save storage because they attempt to replace data patterns that span multiple columns with shorter symbol strings. Value compression can offer savings if you have many rows with columns that contain the same value, such as a city or country name, or if you have columns that contain the default value for the data type of the column.

Procedure

To create a table that uses compression, issue a CREATE TABLE statement.

- If you want to use adaptive compression, include the COMPRESS YES ADAPTIVE clause.
- If you want to use classic row compression, include the COMPRESS YES STATIC clause.
- If you want to use value compression, include the VALUE COMPRESSION clause. If you want to compress data that represents system default column values, also include the COMPRESS SYSTEM DEFAULT clause.

Results

After you create the table, all data that you add to the table from that point in time on is compressed. Any indexes that are associated with the table are also compressed, unless you specify otherwise by using the COMPRESS NO clause of the CREATE INDEX or ALTER INDEX statements.

Examples

Example 1: The following statement creates a table for customer information with adaptive compression enabled. In this example, the table is compressed by using both table-level and page-level compression dictionaries.

```
CREATE TABLE CUSTOMER
     (CUSTOMERNUM      INTEGER,
      CUSTOMERNAME     VARCHAR(80),
      ADDRESS          VARCHAR(200),
      CITY             VARCHAR(50),
      COUNTRY          VARCHAR(50),
      CODE             VARCHAR(15),
      CUSTOMERNUMDIM   INTEGER)
      COMPRESS YES ADAPTIVE;
```

Example 2: The following statement creates a table for customer information with classic row compression enabled. In this example, the table is compressed by using only a table-level compression dictionary.

```
CREATE TABLE CUSTOMER
     (CUSTOMERNUM      INTEGER,
      CUSTOMERNAME     VARCHAR(80),
      ADDRESS          VARCHAR(200),
      CITY             VARCHAR(50),
      COUNTRY          VARCHAR(50),
      CODE             VARCHAR(15),
      CUSTOMERNUMDIM   INTEGER)
      COMPRESS YES STATIC;
```

Example 3: The following statement creates a table for employee salaries. The SALARY column has a default value of 0, and row compression and system default compression are specified for the column.

```
CREATE TABLE EMPLOYEE_SALARY
     (DEPTNO   CHAR(3)      NOT NULL,
      DEPTNAME VARCHAR(36)  NOT NULL,
      EMPNO    CHAR(6)      NOT NULL,
      SALARY   DECIMAL(9,2) NOT NULL WITH DEFAULT COMPRESS SYSTEM DEFAULT)
      COMPRESS YES ADAPTIVE;
```

Note that the VALUE COMPRESSION clause was omitted from this statement. This statement creates a table that is called EMPLOYEE_SALARY; however, a warning message is returned:

```
SQL20140W  COMPRESS column attribute ignored because VALUE COMPRESSION is
deactivated for the table.  SQLSTATE=01648
```

In this case, the COMPRESS SYSTEM DEFAULT clause is not applied to the SALARY column.

Example 4: The following statement creates a table for employee salaries. The SALARY column has a default value of 0, and row compression and system default compression are enabled for the column.

```
CREATE TABLE EMPLOYEE_SALARY
     (DEPTNO   CHAR(3)      NOT NULL,
      DEPTNAME VARCHAR(36)  NOT NULL,
      EMPNO    CHAR(6)      NOT NULL,
      SALARY   DECIMAL(9,2) NOT NULL WITH DEFAULT COMPRESS SYSTEM DEFAULT)
      VALUE COMPRESSION COMPRESS YES ADAPTIVE;
```

In this example, the VALUE COMPRESSION clause is included in the statement, which compresses the default value for the SALARY column.

Enabling compression in an existing table

By using the ALTER TABLE statement, you can modify an existing table to take advantage of the storage-saving benefits of compression.

Before you begin

You must decide which type of compression you want to use: adaptive compression, classic row compression, value compression, or a combination of value compression with either of the two types of row compression. Adaptive compression and classic row compression almost always save storage because they attempt to replace data patterns that span multiple columns with shorter symbol strings. Value compression can offer savings if you have many rows with columns that contain the same value, such as a city or country name, or if you have columns that contain the default value for the data type of the column.

Procedure

To enable compression in an existing table:

1. Issue the ALTER TABLE statement.
   - If you want to use adaptive compression, include the COMPRESS YES ADAPTIVE clause.
   - If you want to use classic row compression, include the COMPRESS YES STATIC clause.
   - If you want to use value compression, include the ACTIVATE VALUE COMPRESSION clause for each column that contains a value you want compressed. If you want to compress data in columns that contain system default values, also include the COMPRESS SYSTEM DEFAULT clause.

   All rows that you subsequently append, insert, load, or update use the new compressed format.
2. Optional: To immediately apply compression to all the existing rows of a table, perform a table reorganization by using the **REORG TABLE** command. If you do not apply compression to all rows at this point, uncompressed rows will not be stored in the new compressed format until the next time that you update them, or the next time the REORG TABLE command runs.

Examples

Example 1: The following statement applies adaptive compression to an existing table that is named CUSTOMER:

```
ALTER TABLE CUSTOMER COMPRESS YES ADAPTIVE
```

Example 2: The following statement applies classic row compression to an existing table that is named CUSTOMER:

```
ALTER TABLE CUSTOMER COMPRESS YES STATIC
```

Example 3: The following statements apply row, value, and system default compression to the SALARY column of an existing table that is named EMPLOYEE_SALARY. The table is then reorganized.

```
ALTER TABLE EMPLOYEE_SALARY
ALTER SALARY COMPRESS SYSTEM DEFAULT
COMPRESS YES ACTIVATE VALUE COMPRESSION;

REORG TABLE EMPLOYEE_SALARY
```

Changing or disabling compression for a compressed table

You can change how a table is compressed or disable compression entirely for a table that has adaptive, classic row, or value compression enabled by using one or more of the various compression-related clauses of the ALTER TABLE statement.

About this task

If you deactivate adaptive or classic row compression, index compression is not affected. If you want to uncompress an index, you must use the ALTER INDEX statement.

Procedure

To deactivate compression for a table, or to change from one type of row compression to another:

1. Issue an ALTER TABLE statement.
   - If you want to deactivate adaptive or classic row compression, include the COMPRESS NO clause.
   - If you want to change to a different type of row compression, specify the type of compression you want using the COMPRESS YES ADAPTIVE or COMPRESS YES STATIC clauses. For example, if you have a table that currently uses classic row compression, and you want to change to adaptive compression, execute the ALTER TABLE statement with the COMPRESS YES ADAPTIVE clause
   - If you want to deactivate value compression, include the DEACTIVATE VALUE COMPRESSION clause.
   - If you want to deactivate the compression of system default values, include the COMPRESS OFF option for the ALTER *column name* clause.
2. Perform an offline table reorganization using the **REORG TABLE** command.

Results

- If you turned off row compression using the COMPRESS NO clause, all row data is uncompressed.
- If you changed from one type of row compression to another, the entire table is compressed using the type of row compression you specified in the ALTER TABLE statement. (See Example 2.)
- Deactivating value compression has the following effects:
  - If a table had columns with COMPRESS SYSTEM DEFAULT enabled, compression is no longer enabled for these columns.
  - Uncompressed columns might cause the row size to exceed the maximum that the current page size of the current table space allows. If this occurs, error message SQL0670N is returned.

Examples

Example 1: Turning off row compression: The following statements turn off adaptive or classic row compression in an table named CUSTOMER and then reorganizes the table to uncompress that data that was previously compressed:

```
ALTER TABLE CUSTOMER COMPRESS NO
REORG TABLE CUSTOMER
```

Example 2: Changing from static to adaptive compression: Assumes that the SALES table currently uses classic row compression. The following statements change the type of compression used to adaptive compression:

```
ALTER TABLE SALES COMPRESS ADAPTIVE YES
REORG TABLE SALES
```

Compression dictionaries

The database manager creates a table-level compression dictionary for each table that you enable for either adaptive or classic row compression. For tables that you enable for adaptive compression, the database manager also creates page-level compression dictionaries.

Both types of dictionaries are used to map repeated byte patterns from table rows to much smaller symbols; these symbols then replace the longer byte patterns in the table rows.

Table-level compression dictionaries

To build table-level dictionaries, the table is scanned for repeating patterns. Entire rows, not just certain fields or parts of rows, are examined for repeating entries or patterns. After collecting the repetitive entries, the database manager builds a compression dictionary, assigning short, numeric keys to those entries. Generally speaking, text strings provide greater opportunities for compression than numeric data; compressing numeric data involves replacing one number with another. Depending on the size of the numbers being replaced, the storage savings might not be as significant as those achieved by compressing text.

When a table-level dictionary is first created, it is built using a sample of data in the table. The dictionary is not updated again unless you explicitly cause the dictionary to be rebuilt using a classic, offline table reorganization. Even if you rebuild the dictionary, the dictionary reflects only a sample of the data from the entire table.

Remember: The table-level dictionary is static; unless you manually rebuild it, it does not change after it is initially created. Even if you do rebuild it, because of the sampling techniques used to create it, the dictionary might not reflect strings that recur within a single page.

The table-level compression dictionary is stored in hidden rows in the same object that they apply to and is cached in memory for quick access. This dictionary does not occupy much space. Even for extremely large tables, the compression dictionary typically occupies only approximately 100 KB.

Page-level compression dictionaries

Adaptive compression uses page-level dictionaries in addition to table-level dictionaries. However, unlike table-level dictionaries, page-level dictionaries are automatically created or recreated as pages are filled by the database manager. Like table-level compression dictionaries, page-level dictionaries are also stored in hidden rows within the table.

Table-level compression dictionary creation

Table-level compression dictionaries for tables that you enable for adaptive or classic row compression can be built automatically or manually. Tables that you enable for adaptive compression include page-level data dictionaries, which are always automatically created.

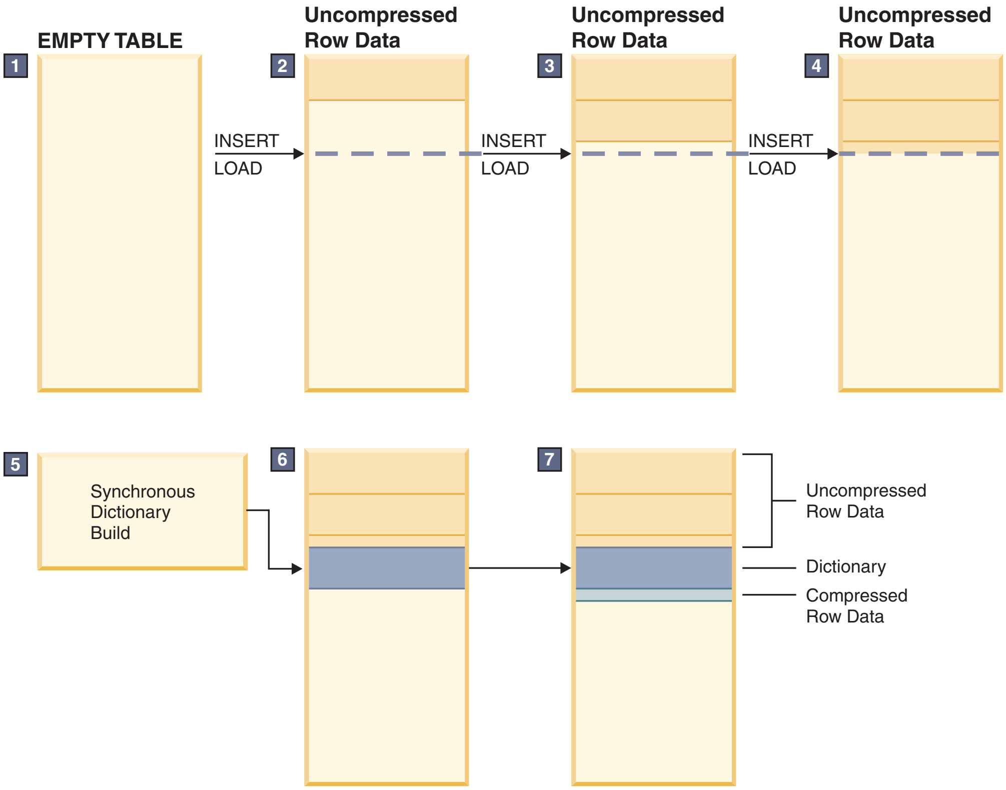
Automatic dictionary creation

Starting with DB2 Version 9.5, a table-level compression dictionary is created automatically if each of the following conditions is met:

- You set the COMPRESS attribute for the table by using the CREATE TABLE or ALTER TABLE statement with the COMPRESS YES ADAPTIVE or COMPRESS YES STATIC clause.
- A table-level compression dictionary does not already exist for the table.
- The table contains sufficient data for constructing a dictionary of repeated data.

Data that you move into the table after the dictionary is created is compressed using the dictionary if compression remains enabled.

The following diagram shows the process by which the compression dictionary is automatically created:



The sequence of events illustrated in the diagram is as follows:

1. A compression dictionary is not yet created, because the table is empty.
2. Data is inserted into the table by using insert or load operations and remains uncompressed.

3. As more data is inserted or loaded into the table, the data remains uncompressed.
4. After a threshold is reached, dictionary creation is triggered automatically if the COMPRESS attribute is set to YES ADAPTIVE or YES STATIC.
5. The dictionary is created.
6. The dictionary is appended to the table.
7. From this point forward, table-level compression is enabled, and the rows that are later inserted or added are compressed by the table-level compression dictionary.

Important: The rows that existed in a table before the dictionary was created remain uncompressed unless you change them or manually rebuild the dictionary.

If you create a table with DB2 Version 9.7 or later and the table contains at least one column of type XML, a second compression dictionary is created. This dictionary is used to compress the XML data that is stored in the default XML storage object that is associated with the table. Compression dictionary creation for XML data occurs automatically if each of the following conditions is met:

- You set the COMPRESS attribute on the table to YES ADAPTIVE or YES STATIC.
- A compression dictionary does not exist within that XML storage object.
- There is sufficient data in the XML storage object.

Restriction: You cannot compress data in XML columns that you created with DB2 Version 9.5 or DB2 Version 9.1. However, you can compress inline XML columns that you add to a table using DB2 Version 9.7 or later, provided the table was created without XML columns in an earlier release of the product. If a table that you created in an earlier release already has one or more XML columns and you want to add a compressed XML column by using DB2 Version 9.7 or later, you must use the ADMIN_MOVE_TABLE stored procedure to migrate the table before you can use compression.

The mechanism for creating table-level compression dictionaries for temporary tables is similar to the mechanism that is used for permanent tables. However, the database manager automatically makes the determination whether to use classic row compression for temporary tables, based on factors such as query complexity and the size of the result set.

Manual dictionary creation

Although dictionaries are created automatically when compression-enabled tables grow to a sufficient size, you can also force a table-level compression dictionary to be created if none exists by using the **REORG TABLE** command with the **RESETDICTIONARY** parameter. This command forces the creation of a compression dictionary if there is at least one row of data in the table. Table reorganization is an offline operation; one benefit of using automatic dictionary creation is that the table remains online as the dictionary is built.

Instead of using the **REORG TABLE** command to force the creation of a new dictionary, you can also use the **INSPECT** command with the **ROWCOMPESTIMATE** parameter. This command creates a compression dictionary if the table does not already have one. The advantage of this approach over performing a table reorganization is that the table remains online. Rows that you add later are subject to compression; however, rows that existed before you ran the **INSPECT** command remain uncompressed until you perform a table reorganization. However, if compression is enabled, automatic dictionary creation will usually take place shortly after you activate compression, likely before you even have a chance to use the **INSPECT** command.

Resetting compression dictionaries

Whether a table-level compression dictionary is created automatically or manually, the dictionary is static; after it is built, it does not change. As you add or update rows, they are compressed based on the data that exists in the compression dictionary. For many situations, this behavior is appropriate. Consider, for example, a table in a database that is used for maintaining customer accounts for a city water utility. Such a table might have columns such as STREET_ADDRESS, CITY, PROVINCE, TELEPHONE_NUM, POSTAL_CODE, and ACCOUNT_TYPE. If a compression dictionary is built with data from a such table, even if it is only a modestly sized table, there is likely sufficient repetitive information for classic row compression to yield significant space savings. Much of the data might be common from customer to customer, for example, the values of the CITY, POSTAL_CODE, or PROVINCE column or portions of the value in the STREET_ADDRESS or TELEPHONE_NUM column.

However, other tables might change significantly over time. Consider a table that is used for retail sales data as follows:

- A master table is used to accumulate data on a month-by-month basis.
- Each month, a new set of records is loaded into the table.

In this case, a compression dictionary created in, for example, April might not reflect repeating data from sales in later parts of the year. In situations where data in a table changes significantly over time, you might want to reset your compression dictionaries by using the **REORG TABLE** command with the **RESETDICTIONARY** parameter. The advantage of resetting the compression dictionary is that data from the entire table is considered when the dictionary is built.

Impact of classic table reorganization on table-level compression dictionaries

When you reorganize a table that you enabled for adaptive compression or classic row compression using classic, offline table reorganization, you can retain the table-level compression dictionary or force the database manager to create a new one.

In DB2 Version 9.5 and later, a table-level compression dictionary is automatically created for a table that you enable for adaptive or classic row compression by using the CREATE TABLE or ALTER TABLE statement with the **COMPRESS YES** subclause. For a new table, the database manager waits until the table grows to a minimal size before creating the dictionary. For an existing table, the compression dictionary is created when the table grows to a sufficient size to allow pattern repetition to become apparent. Compression is applied only to rows that you insert or update after enabling compression.

If you reorganize a table with a classic table reorganization, and a table-level compression dictionary exists, the **KEEPDICTIONARY** parameter of the **REORG TABLE** command is applied implicitly, which retains the dictionary. When you perform the reorganization, all the rows that are processed are subject to compression using that dictionary. If a compression dictionary does not exist and if the table is large enough, a compression dictionary is created, and the rows are subject to compression using that dictionary.

You can force a new table-level compression dictionary to be built by performing a classic table reorganization that uses the **RESETDICTIONARY** parameter of the **REORG TABLE** command. When you specify the **RESETDICTIONARY** parameter, a new compression dictionary is built if there is at least one row in the table, replacing any existing dictionary.

Note: Table-level dictionaries can be rebuilt using only classic table reorganization. If you attempt to perform an inplace table reorganization of a table that has any rows compressed using a page-level compression dictionary, the REORG command fails with a SQL2219 error.

Multiple compression dictionaries for replication source tables

You can combine the DATA CAPTURE CHANGES clause with the COMPRESS YES STATIC or COMPRESS YES ADAPTIVE option for the CREATE TABLE and ALTER TABLE statements to enable row compression on source tables for replication.

When you enable compression, if you also specify the DATA CAPTURE CHANGES clause as part of the commands **REORG TABLE** or **LOAD REPLACE**, a source table can have two table-level compression dictionaries: an active *table-level compression dictionary* and a *historical compression dictionary*. In other words, if DATA CAPTURE CHANGES is enabled, the table-level compression dictionary is not replaced when you run the **REORG TABLE** or **LOAD REPLACE** commands. Instead, a new dictionary is generated, and the previous dictionary is retained.

The historical compression dictionary makes it possible for the db2ReadLog API to extract the row contents in log records that were written before the active dictionary was rebuilt as a result of specifying the **RESETDICTIONARY** option with a **REORG TABLE** or **LOAD** command.

Note: To have log readers return the data within log records in an uncompressed format instead of a raw compressed format, you must set the **iFilterOption** parameter of the db2ReadLog API to DB2READLOG_FILTER_ON.

If you specified the DATA CAPTURE NONE option as part of the CREATE TABLE statement used to create the table, then issuing the **REORG TABLE** command or performing table truncate operations by issuing the **LOAD REPLACE**, **IMPORT REPLACE**, or **TRUNCATE TABLE** command removes the historical compression dictionary for the table.

To see whether there is a historical dictionary present for the table, check the HISTORICAL_DICTIONARY column in the result set of the ADMIN_GET_TAB_DICTIONARY_INFO table function.

Table partitioning and data organization schemes

Table partitioning is a data organization scheme in which table data is divided across multiple data partitions according to values in one or more partitioning columns of the table. Data from a given table is partitioned into multiple storage objects, which can be in different table spaces.

For complete details about table partitioning and data organization schemes, see the *Partitioning and Clustering Guide*.

Creating tables

The database manager controls changes and access to the data stored in the tables. You can create tables by using the CREATE TABLE statement.

Complex statements can be used to define all the attributes and qualities of tables. However, if all the defaults are used, the statement to create a table is simple.

Declaring temporary tables

To define temporary tables from within your applications, use the DECLARE GLOBAL TEMPORARY TABLE statement.

About this task

Temporary tables, also referred to as user-defined temporary tables, are used by applications that work with data in the database. Results from manipulation of the data need to be stored temporarily in a table. A user temporary table space must exist before declaring temporary tables.

Note: The description of temporary tables does not appear in the system catalog thus making it not persistent for, and not able to be shared with, other applications. When the application using this table terminates or disconnects from the database, any data in the table is deleted and the table is implicitly dropped. Temporary tables do not support:

- User-defined type columns
- LONG VARCHAR columns
- XML columns for created global temporary tables

Example

```
DECLARE GLOBAL TEMPORARY TABLE temptbl
   LIKE empltabl
   ON COMMIT DELETE ROWS
   NOT LOGGED
   IN usr_tbsp
```

This statement defines a temporary table called `temptbl`. This table is defined with columns that have exactly the same name and description as the columns of the `empltabl`. The implicit definition only includes the column name, data type, nullability characteristic, and column default value attributes. All other column attributes including unique constraints, foreign key constraints, triggers, and indexes are not defined. With ON COMMIT DELETE ROWS (any DELETE ROWS option), the database manager always deletes rows whether there's a cursor with a HOLD open on the table or not. The database manager optimizes a NOT LOGGED delete by implementing an internal TRUNCATE, if no WITH HOLD cursors are open, otherwise, the database manager deletes the rows one at a time.

The table is dropped implicitly when the application disconnects from the database. For more information, see the DECLARE GLOBAL TEMPORARY TABLE statement.

Creating and connecting to created temporary tables

Created temporary tables are created using the CREATE GLOBAL TEMPORARY TABLE statement. The first time an application refers to a created temporary table using a connection, a private version of the created temporary table is instantiated for use by the application using the connection.

About this task

Similar to declared temporary tables, created temporary tables are used by applications that work with data in the database, where the results from manipulation of the data need to be stored temporarily in a table. Whereas declared temporary table information *is not* saved in the system catalog tables, and must be defined in every session where it is used, created temporary table information *is* saved in the system catalog and is not required to be defined in every session where it is used, thus making it persistent and able to be shared with other applications, across different connections. A user temporary table space must exist before created temporary tables can be created.

Note: The first implicit or explicit reference to the created temporary table that is executed by any program using the connection creates an empty instance of the given created temporary table. Each connection that references this created temporary table has its own unique instance of the created temporary table, and the instance is not persistent beyond the life of the connection.

References to the created temporary table name in multiple connections refer to the same, single, persistent created temporary table definition, and to a distinct instance of the created temporary table for each connection at the current server. If the created temporary table name that is being referenced is not qualified, it is implicitly qualified using the standard qualification rules that apply to SQL statements.

The owner implicitly has all table privileges on the created temporary table, including the authority to drop it. The owner's table privileges can be granted and revoked, either individually or with the ALL clause. Another authorization ID can access the created temporary table only if it has been granted appropriate privileges.

Indexes and SQL statements that modify data (such as INSERT, UPDATE, and DELETE) are supported. Indexes can only be created in the same table space as the created temporary table.

For the CREATE GLOBAL TEMPORARY TABLE statement: locking and recovery do not apply; logging applies only when the LOGGED clause is specified. For more options, see the CREATE GLOBAL TEMPORARY statement.

Created temporary tables cannot be:
- Associated with security policies
- Table partitioned
- Multidimensional clustering (MDC) tables
- Insert time clustering (ITC) tables
- Range-clustered (RCT)
- Distributed by replication

Materialized query tables (MQTs) cannot be created on created temporary tables.

Created temporary tables do not support the following column types, object types, and table or index operations:

- XML columns
- Structured types
- Referenced types
- Constraints
- Index extensions
- LOAD
- LOAD TABLE
- ALTER TABLE
- RENAME TABLE
- RENAME INDEX
- REORG TABLE
- REORG INDEX
- LOCK TABLE

For more information, see the CREATE GLOBAL TEMPORARY TABLE statement.

Example

```
CREATE GLOBAL TEMPORARY TABLE temptbl
   LIKE empltabl
   ON COMMIT DELETE ROWS
   NOT LOGGED
   IN usr_tbsp
```

This statement creates a temporary table called `temptbl`. This table is defined with columns that have exactly the same name and description as the columns of the `empltabl`. The implicit definition only includes the column name, data type, nullability characteristic, and column default value attributes of the columns in `empltab1`. All other column attributes including unique constraints, foreign key constraints, triggers, and indexes are not implicitly defined.

A COMMIT always deletes the rows from the table. If there are any HOLD cursors open on the table, they can be deleted using TRUNCATE statement, which is faster, but will “normally” have to be deleted row by row. Changes made to the temporary table are not logged. The temporary table is placed in the specified user temporary table space, `usr tbsp`. This table space must exist or the creation of this table will fail.

When an application that instantiated a created temporary table disconnects from the database, the application's instance of the created temporary table is dropped.

Distinctions between DB2 base tables and temporary tables

DB2 base tables and the two types of temporary tables have several distinctions.

The following table summarizes important distinctions between base tables, created temporary tables, and declared temporary tables.

Table 21. Important distinctions between DB2 base tables and DB2 temporary tables

Area of distinction	Distinction
Creation, persistence, and ability to share table descriptions	**Base tables:** The CREATE TABLE statement puts a description of the table in the catalog view SYSCAT.TABLES. The table description is persistent and is shareable across different connections. The name of the table in the CREATE TABLE statement can be qualified. If the table name is not qualified, it is implicitly qualified using the standard qualification rules applied to SQL statements.
	Created temporary tables: The CREATE GLOBAL TEMPORARY TABLE statement puts a description of the table in the catalog view SYSCAT.TABLES. The table description is persistent and is shareable across different connections. The name of the table in the CREATE GLOBAL TEMPORARY TABLE statement can be qualified. If the table name is not qualified, it is implicitly qualified using the standard qualification rules applied to SQL statements.
	Declared temporary tables: The DECLARE GLOBAL TEMPORARY TABLE statement does not put a description of the table in the catalog. The table description is not persistent beyond the life of the connection that issued the DECLARE GLOBAL TEMPORARY TABLE statement and the description is known only to that connection. Thus, each connection could have its own possibly unique description of the same declared temporary table. The name of the table in the DECLARE GLOBAL TEMPORARY TABLE statement can be qualified. If the table name is qualified, SESSION must be used as the schema qualifier. If the table name is not qualified, SESSION is implicitly used as the qualifier.
Table instantiation and ability to share data	**Base tables:** The CREATE TABLE statement creates one empty instance of the table, and all connections use that one instance of the table. The table and data are persistent.
	Created temporary tables: The CREATE GLOBAL TEMPORARY TABLE statement does not create an instance of the table. The first implicit or explicit reference to the table in an open, select, insert, update, or delete operation that is executed by any program using the connection creates an empty instance of the given table. Each connection that references the table has its own unique instance of the table, and the instance is not persistent beyond the life of the connection.
	Declared temporary tables: The DECLARE GLOBAL TEMPORARY TABLE statement creates an empty instance of the table for the connection. Each connection that declares the table has its own unique instance of the table, and the instance is not persistent beyond the life of the connection.

Table 21. Important distinctions between DB2 base tables and DB2 temporary tables (continued)

Area of distinction	Distinction
References to the table during the connection	**Base tables:** References to the table name in multiple connections refer to the same single persistent table description and to the same instance at the current server. If the table name that is being referenced is not qualified, it is implicitly qualified using the standard qualification rules that apply to SQL statements.
	Created temporary tables: References to the table name in multiple connections refer to the same single persistent table description but to a distinct instance of the table for each connection at the current server. If the table name that is being referenced is not qualified, it is implicitly qualified using the standard qualification rules that apply to SQL statements.
	Declared temporary tables: References to the table name in multiple connections refer to a distinct description and instance of the table for each connection at the current server. References to the table name in an SQL statement (other than the DECLARE GLOBAL TEMPORARY TABLE statement) must include SESSION as the schema qualifier. If the table name is not qualified with SESSION, the reference is assumed to be to a base table.
Table privileges and authorization	**Base tables:** The owner implicitly has all table privileges on the table and the authority to drop the table. The owner's table privileges can be granted and revoked, either individually or with the ALL clause. Another authorization ID can access the table only if it has been granted appropriate privileges for the table.
	Created temporary tables: The owner implicitly has all table privileges on the table and the authority to drop the table. The owner's table privileges can be granted and revoked, either individually or with the ALL clause. Another authorization ID can access the table only if it has been granted appropriate privileges for the table.
	Declared temporary tables: PUBLIC implicitly has all table privileges on the table without GRANT authority and also has the authority to drop the table. These table privileges cannot be granted or revoked. Any authorization ID can access the table without requiring a grant of any privileges for the table.
Indexes and other SQL statement support	**Base tables:** Indexes and SQL statements that modify data (INSERT, UPDATE, DELETE, and so on) are supported. Indexes can be in different table spaces.
	Created temporary tables: Indexes and SQL statements that modify data (INSERT, UPDATE, DELETE, and so on) are supported. Indexes can only be in the same table space as the table.
	Declared temporary tables: Indexes and SQL statements that modify data (INSERT, UPDATE, DELETE, and so on) are supported. Indexes can only be in the same table space as the table.

Table 21. Important distinctions between DB2 base tables and DB2 temporary tables (continued)

Area of distinction	Distinction
Locking, logging, and recovery	**Base tables:** Locking, logging, and recovery do apply.
	Created temporary tables: Locking and recovery do not apply, however logging does apply when LOGGED is explicitly specified. Undo recovery (rolling back changes to a savepoint or the most recent commit point) is supported when only when LOGGED is *explicitly* specified.
	Declared temporary tables: Locking and recovery do not apply, however logging only applies when LOGGED is explicitly or implicitly specified. Undo recovery (rolling back changes to a savepoint or the most recent commit point) is supported when LOGGED is explicitly or implicitly specified.

Altering tables

When altering tables, there are some useful options to be aware of, such as the ALTER COLUMN SET DATA TYPE option and the unlimited REORG-recommended operations that can be performed within a single transaction.

Alter table SET DATA TYPE support

The ALTER COLUMN SET DATA TYPE option on the ALTER TABLE statement supports all compatible types.

Altering the column data type can cause data loss. Some of this loss is consistent with casting rules; for example, blanks can be truncated from strings without returning an error, and converting a DECIMAL to an INTEGER results in truncation. To prevent unexpected errors, such as overflow errors, truncation errors, or any other kind of error returned by casting, existing column data is scanned, and messages about conflicting rows are written to the notification log. Column default values are also checked to ensure that they conform to the new data type.

If a data scan does not report any errors, the column type is set to the new data type, and the existing column data is cast to the new data type. If an error is reported, the ALTER TABLE statement fails.

Altering a VARCHAR, VARGRAPHIC, or LOB column to a data type that is sooner in the data type precedence list (see the *Promotion of data types* topic) is not supported.

Example

Change the data type of the SALES column in the SALES table from INTEGER to SMALLINT.

```
alter table sales alter column sales set data type smallint
DB20000I  The SQL command completed successfully.
```

Change the data type of the REGION column in the SALES table from VARCHAR(15) to VARCHAR(14).

```
alter table sales alter column region set data type varchar(14)
...
SQL0190N  ALTER TABLE "ADMINISTRATOR.SALES" specified attributes for column
"REGION" that are not compatible with the existing column.  SQLSTATE=42837
```

Change a column type in a base table. There are views and functions that are directly or indirectly dependent on the base table.

```
create table t1 (c1 int, c2 int);

create view v1 as select c1, c2 from t1;
create view v2 as select c1, c2 from v1;

create function foo1 ()
 language sql
 returns int
 return select c2 from t1;

create view v3 as select c2 from v2
 where c2 = foo1();

create function foo2 ()
 language sql
 returns int
 return select c2 from v3;

alter table t1
 alter column c1
  set data type smallint;

select * from v2;
```

The ALTER TABLE statement, which down casts the column type from INTEGER to SMALLINT, invalidates v1, v2, v3, and foo2. Under revalidation deferred semantics, `select * from v2` successfully revalidates v1 and v2, and the c1 columns in both v1 and v2 are changed to SMALLINT. But v3 and foo2 are not revalidated, because they are not referenced after being invalidated, and they are above v2 in the dependency hierarchy chain. Under revalidation immediate semantics, the ALTER TABLE statement revalidates all the dependent objects successfully.

Multiple ALTER TABLE operations within a single unit of work

Certain ALTER TABLE operations, like dropping a column, altering a column type, or altering the nullability property of a column may put the table into a reorg pending state. In this state, many types of queries cannot be run; you must perform a table reorganization before the table becomes available for some types of queries. However, even with the table in a reorg pending state, you can still perform multiple ALTER TABLE operations before doing a reorg.

Beginning with DB2 Version 9.7, you can perform an unlimited number of ALTER TABLE statements within a single *unit of work*. However, after three units of work have been performed that include such operations, a REORG TABLE command must be run.

Adding and dropping columns

To add columns to existing tables, or to drop columns from existing tables, use the ALTER TABLE statement with the ADD COLUMN or DROP COLUMN clause. The table must not be a typed table.

About this task

For all existing rows in the table, the value of the new column is set to its default value. The new column is the last column of the table; that is, if initially there are *n* columns, the added column is column *n*+1. Adding the new column must not make the total byte count of all columns exceed the row size limit.

Procedure

- To add a column, issue the ALTER TABLE statement with the ADD COLUMN clause. For example:

```
ALTER TABLE SALES
  ADD COLUMN SOLD_QTY
  SMALLINT NOT NULL DEFAULT 0
```

- To delete or drop a column, issue the ALTER TABLE statement with the DROP COLUMN clause. For example:

```
ALTER TABLE SALES
  DROP COLUMN SOLD_QTY
```

Modifying DEFAULT clause column definitions

The DEFAULT clause provides a default value for a column in the event that a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a specific default value is not specified following the DEFAULT keyword, the default value depends on the data type. If a column is defined as an XML or structured type, then a DEFAULT clause cannot be specified.

About this task

Omission of DEFAULT from a column-definition results in the use of the null value as the default for the column, as described in: “Default column and data type definitions” on page 209.

Specific types of values that can be specified with the DEFAULT keyword, see the ALTER TABLE statement.

Modifying the generated or identity property of a column

You can add and drop the generated or identity property of a column in a table using the ALTER COLUMN clause in the ALTER TABLE statement.

You can do one of the following actions:

- When working with an existing non-generated column, you can add a generated expression attribute. The modified column then becomes a generated column.
- When working with an existing generated column, you can drop a generated expression attribute. The modified column then becomes a normal, non-generated column.
- When working with an existing non-identity column, you can add a identity attribute. The modified column then becomes an identity column.
- When working with an existing identity column, you can drop the identity attribute. The modified column then becomes a normal, non-generated, non-identity column.
- When working with an existing identity column, you can alter the column from being GENERATED ALWAYS to GENERATED BY DEFAULT. The reverse is also

true; that is, you can alter the column from being GENERATED BY DEFAULT to GENERATED ALWAYS. This is only possible when working with an identity column.

- You can drop the default attribute from the user-defined default column. When you do this, the new default value is null.
- You can drop the default, identity, or generation attribute and then set a new default, identity, or generation attribute in the same ALTER COLUMN statement.
- For both the CREATE TABLE and ALTER TABLE statements, the "ALWAYS" keyword is optional in the generated column clause. This means that GENERATED ALWAYS is equivalent to GENERATED.

Modifying column definitions

Use the ALTER TABLE statement to drop columns, or change their types and attributes. For example, you can increase the length of an existing VARCHAR or VARGRAPHIC column. The number of characters might increase up to a value dependent on the page size used.

About this task

To modify the default value associated with a column, once you have defined the new default value, the new value is used for the column in any subsequent SQL operations where the use of the default is indicated. The new value must follow the rules for assignment and have the same restrictions as documented under the CREATE TABLE statement.

Note: Generate columns cannot have their default value altered by this statement.

When changing these table attributes using SQL, it is no longer necessary to drop the table and then re-create it, a time consuming process that can be complex when object dependencies exist.

Procedure

- To modify the length and type of a column of an existing table using the command line, enter:

  ```
  ALTER TABLE table_name
     ALTER COLUMN column_name
     modification_type
  ```

  For example, to increase a column up to 4000 characters, use something similar to the following:

  ```
  ALTER TABLE t1
     ALTER COLUMN colnam1
     SET DATA TYPE VARCHAR(4000)
  ```

  In another example, to allow a column to have a new VARGRAPHIC value, use a statement similar to the following:

  ```
  ALTER TABLE t1
     ALTER COLUMN colnam2
     SET DATA TYPE VARGRAPHIC(2000)
  ```

  You cannot alter the column of a typed table. However, you can add a scope to an existing reference type column that does not already have a scope defined. For example:

  ```
  ALTER TABLE t1
     ALTER COLUMN colnamt1
     ADD SCOPE typtab1
  ```

- To modify a column to allow for LOBs to be included inline, enter:

  ```
  ALTER TABLE table_name
     ALTER COLUMN column_name
     SET INLINE LENGTH new_LOB_length
  ```

  For example, if you want LOBs of 1000 bytes or less to be included in a base table row, use a statement similar to the following:

  ```
  ALTER TABLE t1
     ALTER COLUMN colnam1
     SET INLINE LENGTH 1004
  ```

  In this case, the length is set to 1004, rather than 1000. This is because inline LOBs require an additional 4 bytes of storage over and above the size of the LOB itself.
- To modify the default value of a column of an existing table using the command line, enter:

  ```
  ALTER TABLE table_name
     ALTER COLUMN column_name
     SET DEFAULT 'new_default_value'
  ```

  For example, to change the default value for a column, use something similar to the following:

  ```
  ALTER TABLE t1
     ALTER COLUMN colnam1
     SET DEFAULT '123'
  ```

Altering materialized query table properties

With some restrictions, you can change a materialized query table to a regular table or a regular table to a materialized query table. You cannot change other table types; only regular and materialized query tables can be changed. For example, you cannot change a replicated materialized query table to a regular table, nor the reverse.

About this task

Once a regular table has been altered to a materialized query table, the table is placed in a set integrity pending state. When altering in this way, the `fullselect` in the materialized query table definition must match the original table definition, that is:

- The number of columns must be the same.
- The column names and positions must match.
- The data types must be identical.

If the materialized query table is defined on an original table, then the original table cannot itself be altered into a materialized query table. If the original table has triggers, check constraints, referential constraints, or a defined unique index, then it cannot be altered into a materialized query table. If altering the table properties to define a materialized query table, you are not allowed to alter the table in any other way in the same ALTER TABLE statement.

When altering a regular table into a materialized query table, the `fullselect` of the materialized query table definition cannot reference the original table directly or indirectly through views, aliases, or materialized query tables.

To change a materialized query table to a regular table, use the following command:

```
ALTER TABLE sumtable
   SET SUMMARY AS DEFINITION ONLY
```

To change a regular table to a materialized query table, use the following command:

```
 ALTER TABLE regtable
   SET SUMMARY AS <fullselect>
```

The restrictions on the `fullselect` when altering the regular table to a materialized query table are very much like the restrictions when creating a summary table using the CREATE SUMMARY TABLE statement.

Refreshing the data in a materialized query table

You can refresh the data in one or more materialized query tables by using the REFRESH TABLE statement. The statement can be embedded in an application program, or issued dynamically. To use this statement, you must have DATAACCESS authority, or CONTROL privilege on the table to be refreshed.

Example

The following example shows how to refresh the data in a materialized query table:

```
REFRESH TABLE SUMTAB1
```

Renaming tables and columns

You can use the RENAME statement to rename an existing table. To rename columns, use the ALTER TABLE statement.

About this task

When renaming tables, the source table must not be referenced in any existing definitions (view or materialized query table), triggers, SQL functions, or constraints. It must also not have any generated columns (other than identity columns), or be a parent or dependent table. Catalog entries are updated to reflect the new table name. For more information and examples, see the RENAME statement.

The RENAME COLUMN clause is an option on the ALTER TABLE statement. You can rename an existing column in a base table to a new name without losing stored data or affecting any privileges or label-based access control (LBAC) policies that are associated with the table.

Only the renaming of base table columns is supported. Renaming columns in views, materialized query tables (MQTs), declared and created temporary tables, and other table-like objects is not supported.

Invalidation and revalidation semantics for the rename column operation are similar to those for the drop column operation; that is, all dependent objects are invalidated. Revalidation of all dependent objects following a rename column operation is always done immediately after the invalidation, even if the **`auto_reval`** database configuration parameter is set to DISABLED.

The following example shows the renaming of a column using the ALTER TABLE statement:

```
ALTER TABLE org RENAME COLUMN deptnumb TO deptnum
```

To change the definition of existing columns, see the "Changing column properties" topic or the ALTER TABLE statement.

Viewing table definitions

You can use the SYSCAT.TABLES and SYSCAT.COLUMNS catalog views to view table definitions. For SYSCAT.COLUMNS, each row represents a column defined for a table, view, or nickname. To see the data in the columns, use the SELECT statement.

About this task

You can also use the following views and table functions to view table definitions:

- ADMINTEMPCOLUMNS administrative view
- ADMINTEMPTABLES administrative view
- ADMIN_GET_TEMP_COLUMNS table function - Retrieve column information for temporary tables
- ADMIN_GET_TEMP_TABLES table function - Retrieve information for temporary tables

Dropping tables

A table can be dropped with a DROP TABLE statement.

About this task

When a table is dropped, the row in the SYSCAT.TABLES system catalog view that contains information about that table is dropped, and any other objects that depend on the table are affected. For example:

- All column names are dropped.
- Indexes created on any columns of the table are dropped.
- All views based on the table are marked inoperative.
- All privileges on the dropped table and dependent views are implicitly revoked.
- All referential constraints in which the table is a parent or dependent are dropped.
- All packages and cached dynamic SQL and XQuery statements dependent on the dropped table are marked invalid, and remain so until the dependent objects are re-created. This includes packages dependent on any supertable above the subtable in the hierarchy that is being dropped.
- Any reference columns for which the dropped table is defined as the scope of the reference become “unscoped”.
- An alias definition on the table is not affected, because an alias can be undefined
- All triggers dependent on the dropped table are marked inoperative.

Restrictions

An individual table cannot be dropped if it has a subtable.

Procedure

- To drop a table, use a DROP TABLE statement.

  The following statement drops the table called DEPARTMENT:

  ```
  DROP TABLE DEPARTMENT
  ```
- To drop all the tables in a table hierarchy, use a DROP TABLE HIERARCHY statement.

  The DROP TABLE HIERARCHY statement must name the root table of the hierarchy to be dropped. For example:

  ```
  DROP TABLE HIERARCHY person
  ```

Results

There are differences when dropping a table hierarchy compared to dropping a specific table:

- DROP TABLE HIERARCHY does not activate deletion-triggers that would be activated by individual DROP TABLE statements. For example, dropping an individual subtable would activate deletion-triggers on its supertables.
- DROP TABLE HIERARCHY does not make log entries for the individual rows of the dropped tables. Instead, the dropping of the hierarchy is logged as a single event.

Chapter 20. Time Travel Query using temporal tables

You can use temporal tables to associate time-based state information with your data. Data in tables that do not use temporal support are deemed to be applicable to the present, while data in temporal tables can be valid for a period defined by the database system, user applications, or both.

There are many business needs requiring the storage and maintenance of time-based data. Without this capability in a database, it is expensive and complex to maintain a time-focused data support infrastructure. With temporal tables, the database can store and retrieve time-based data without additional application logic. For example, a database can store the history of a table (deleted rows or the original values of rows that have been updated) so you can query the past state of your data. You can also assign a date range to a row of data to indicate when it is deemed to be valid by your applications or business rules.

A temporal table records the period when a row is valid. A period is an interval of time that is defined by two date or time columns in the temporal table. A period contains a begin column and an end column. The begin column indicates the beginning of the period, and the end column indicates the end of the period. The beginning value of a period is inclusive, while the ending value of a period is exclusive. For example, a row with a period from January 1 to February 1 is valid from January 1, until January 31 at midnight.

Two period types are supported:

System periods
A system period consists of a pair of columns with database manager-maintained values that indicate the period when a row is current. The begin column contains a timestamp value for when a row was created. The end column contains a timestamp value for when a row was updated or deleted. When a system-period temporal table is created, it contains the currently active rows. Each system-period temporal table is associated with a history table that contains any changed rows.

Application periods
An application period consists of a pair of columns with user or application-supplied values that indicate the period when a row is valid. The begin column indicates the time when a row is valid from. The end column indicates the time when a row stops being valid. A table with an application period is called an application-period temporal table.

You can check whether a table has temporal support by querying the SYSCAT.TABLES system catalog view. For example:

```
SELECT TABSCHEMA, TABNAME, TEMPORALTYPE FROM SYSCAT.TABLES
```

The returned values for `TEMPORALTYPE` are defined as follows:

A Application-period temporal table

B Bitemporal table

N Not a temporal table

S System-period temporal table

System-period temporal tables

A system-period temporal table is a table that maintains historical versions of its rows. Use a system-period temporal table to store current versions of your data and use its associated history table to transparently store your updated and deleted data rows.

A system-period temporal table includes a SYSTEM_TIME period with columns that capture the begin and end times when the data in a row is current. The database manager also uses the SYSTEM_TIME period to preserve historical versions of each table row whenever updates or deletes occur. The database manager stores these rows in a history table that is exclusively associated with a system-period temporal table. Adding versioning establishes the link between the system-period temporal table and the history table. With a system-period temporal table, your queries have access to your data at the current point in time and the ability to retrieve data from past points in time.

A system-period temporal table also includes a transaction start-ID column. This column captures the time when execution started for a transaction that impacts the row. If multiple rows are inserted or updated within a single SQL transaction, then the values for the transaction start-ID column are the same for all the rows and are unique from the values generated for this column by other transactions. This common start-ID column value means you can use the transaction start-ID column to identify all the rows in the tables that were written by the same transaction.

History tables

Each system-period temporal table requires a history table. When a row is updated or deleted from a system-period temporal table, the database manager inserts a copy of the old row into its associated history table. This storage of old system-period temporal table data gives you the ability to retrieve data from past points in time.

In order to store row data, the history table columns and system-period temporal table columns must have the same names, order, and data types. You can create a history table with the same names and descriptions as the columns of the system-period temporal table by using the LIKE clause of the CREATE TABLE statement, for example:

```
CREATE TABLE employees_history LIKE employees IN hist_space;
```

An existing table can be used as a history table if it avoids the restrictions listed in the description of the ALTER TABLE statement USE HISTORY clause.

After you create a history table, you add versioning to establish the link between the system-period temporal table and the history table.

```
ALTER TABLE employees ADD VERSIONING USE HISTORY TABLE employees_history;
```

A history table is subject to the following rules and restrictions when versioning is enabled:

- A history table cannot explicitly be dropped. It can only implicitly be dropped when the associated system-period temporal table is dropped.
- History table columns cannot explicitly be added, dropped, or changed.
- A history table must not be defined as parent, child, or self-referencing in a referential constraint. Access to the history table is restricted to prevent cascaded actions to the history table.

- A table space that contains a history table, but not its associated system-period temporal table, cannot be dropped.

You should rarely need to explicitly change a history table. Doing so might jeopardize your ability to audit a system-period temporal table data history. You should restrict access to a history table to protect its data.

Under normal operations, a history table experiences mostly insert and read activities. Updates and deletes are rare. The absence of updates and deletes means that history tables typically do not have free space that can be reused for the inserting of new rows. If row inserts into the history table are negatively impacting workload performance, you can eliminate the search for free space by altering the definition of the history table by using the APPEND ON option. This option avoids the processing associated with free space searches and directly appends new rows to the end of the table.

```
ALTER TABLE employees_history APPEND ON;
```

When a system-period temporal table is dropped, the associated history table and any indexes defined on the history table are implicitly dropped. To avoid losing historical data when a system-period temporal table is dropped, you can either create the history table with the RESTRICT ON DROP attribute or alter the history table by adding the RESTRICT ON DROP attribute.

```
CREATE TABLE employees_history LIKE employees WITH RESTRICT ON DROP;
```

Because history tables experience more inserts than deletes, your history tables are always growing and so are consuming an increasing amount of storage. Deciding how to prune your history tables to get rid of the rows that you no longer need can be a complex task. You need to understand the value of your individual records. Some content, like customer contracts, might be untouchable and can never be deleted. While other records, like website visitor information, can be pruned without concern. Often it is not the age of a row that determines when it can be pruned and archived, but rather it is some business logic that is the deciding factor. The following list contains some possible rules for pruning:

- Prune rows selected by a user-supplied query that reflects business rules.
- Prune rows older than a certain age.
- Prune history rows when more than N versions exist for that record (retain only the latest N versions).
- Prune history rows when the record is deleted from the associated system-period temporal table (when there are no current versions).

There are several ways to periodically prune old data from a history table:

- Use range partitioning and detach old partitions from the history table.
- Use DELETE statements to remove rows from the table. If using DELETE statements, you might observe the following guidelines:
  - Periodically reorganize the history table to release the free space left behind by the delete operations.
  - Ensure that the history table was not altered to use the APPEND ON option, allowing inserts to search for free space.

SYSTEM_TIME period

The SYSTEM_TIME period columns for a system-period temporal table indicate when the version of a row is current.

The SYSTEM_TIME period contains a pair of TIMESTAMP(12) columns whose values are generated by the database manager. The columns must be defined as NOT NULL with an applicable GENERATED ALWAYS AS option. The begin column of the period must be a row-begin column and the end column of the period must be a row-end column.

```
CREATE TABLE policy_info
(
 policy_id       CHAR(4) NOT NULL,
 coverage        INT NOT NULL,
 sys_start   TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
 sys_end  TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
 ts_id   TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID,
 PERIOD SYSTEM_TIME (sys_start, sys_end)
) IN policy_space;
```

Row-begin column

This column represents the time when the row data became current. The database manager generates a value for this column by using a reading of the system clock at the moment it executes the first data change statement in the transaction that generates the row. If multiple rows are inserted or updated within a single SQL transaction, the values for the row-begin column are the same for all the impacted rows. The values for these row-begin columns are unique from the values generated for the row-begin columns for other transactions. A row-begin column is required as the begin column of a SYSTEM_TIME period, which must be defined for each system-period temporal table.

When an existing regular table is altered to make it a system-period temporal table, a row-begin column is added to the table. The row-begin column is populated with a default of `0001-01-01-00.00.00.000000000000` for the TIMESTAMP(12) data type value for any existing rows.

Row-end column

This column represents the time when the row data was no longer current. For rows in a history table, the value in the row-end column represents when the row was added to the history table. The rows in the system-period temporal table are by definition current, so the row-end column is populated with a default value for the TIMESTAMP(12) data type (for example: `9999-12-30-00.00.00.000000000000`). A row-end column is required as the end column of a SYSTEM_TIME period, which must be defined for each system-period temporal table.

When an existing regular table is altered to make it a system-period temporal table, a row-end column is added to the table. The row-end column is populated with the a value for the TIMESTAMP(12) data type (default value: `9999-12-30-00.00.00.000000000000`) for any existing rows. The value is intentionally one day earlier than the maximum possible time stamp to avoid a possible overflow to a 5-digit year when converting the value to a different time zone.

Since row-begin and row-end are generated columns, there is no implicit check constraint generated for SYSTEM_TIME that ensures that the value for an end column is greater than the value for its begin column in a system-period temporal table. This lack of a check constraint differs from an application-period temporal table where there is a check constraint associated with its BUSINESS_TIME. A row where the value for the end column is less than the value for the begin column cannot be returned when a period-specification is used to query the table. You can define a constraint to guarantee that the value for end column is greater than the value for begin column. This guarantee is useful when supporting operations that explicitly input data into these generated columns, such as a load operation.

The **systime_period_adj** database configuration parameter is used to specify what action to take when a history row for a system-period temporal table is generated with an end column value that is less than the value for begin column.

Creating a system-period temporal table

Creating a system-period temporal table results in a table that tracks when data changes occur and preserves historical versions of that data.

About this task

When creating a system-period temporal table, include attributes that indicate when data in a row is current and when transactions affected the data:

- Include row-begin and row-end columns that are used by the SYSTEM_TIME period to track when a row is current.
- Include a transaction start-ID column that captures the start times for transactions that affect rows.
- Create a history table to receive old rows from the system-period temporal table.
- Add versioning to establish the link between the system-period temporal table and the history table.

The row-begin, row-end, and transaction start-ID columns can be defined as IMPLICITLY HIDDEN. Since these columns and their entries are generated by the database manager, hiding them can minimize any potential negative affects on your applications. These columns are then unavailable unless referenced, for example:

- A SELECT * query run against a table does not return any implicitly hidden columns in the result table.
- An INSERT statement does not expect a value for any implicitly hidden columns.
- The LOAD, IMPORT, and EXPORT commands can use the *includeimplicitlyhidden* modifier to work with implicitly hidden columns.

A system-period temporal table can be defined as a parent or a child in a referential constraint. However, the referential constraints are applied only to the current data, that is the data in the system-period temporal table. The constraints are not applied to the associated history table. In order to minimize inconsistencies when a system-period temporal table is a child table in a referential constraint, the parent table should also be a system-period temporal table.

Note: While the row-begin, row-end, and transaction start-ID generated columns are required when creating a system-period temporal table, you can also create a regular table with these generated columns.

The example in the following section shows the creation of a table that stores policy information for the customers of an insurance company.

Procedure

To create a system-period temporal table.

1. Create a table with a SYSTEM_TIME attribute. For example:

```
CREATE TABLE policy_info
(
 policy_id    CHAR(4) NOT NULL,
 coverage     INT NOT NULL,
 sys_start    TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
 sys_end      TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
 ts_id        TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID,
 PERIOD SYSTEM_TIME (sys_start, sys_end)
) IN policy_space;
```

2. Create a history table. For example:

```
CREATE TABLE hist_policy_info
(
 policy_id    CHAR(4) NOT NULL,
 coverage     INT NOT NULL,
 sys_start    TIMESTAMP(12) NOT NULL,
 sys_end      TIMESTAMP(12) NOT NULL,
 ts_id        TIMESTAMP(12) NOT NULL
) IN hist_space;
```

   You can also create a history table with the same names and descriptions as the columns of the system-period temporal table by using the LIKE clause of the CREATE TABLE statement. For example:

```
CREATE TABLE hist_policy_info LIKE policy_info IN hist_space;
```

3. Add versioning to the system-period temporal table to establish a link to the history table. For example:

```
ALTER TABLE policy_info ADD VERSIONING USE HISTORY TABLE hist_policy_info;
```

Results

The policy_info table stores the insurance coverage level for a customer. The SYSTEM_TIME period related columns (sys_start and sys_end) show when a coverage level row is current. The ts_id column lists the time when execution started for a transaction that impacted the row.

Table 22. Created system-period temporal table (policy_info)

policy_id	coverage	sys_start	sys_end	ts_id

The hist_policy_info history table receives the old rows from the policy_info table.

Table 23. Created history table (hist_policy_info)

policy_id	coverage	sys_start	sys_end	ts_id

Example

This section contains more creating system-period temporal table examples.

Hiding columns

The following example creates the policy_info table with the TIMESTAMP(12) columns (sys_start, sys_end and ts_id) marked as implicitly hidden.

```
CREATE TABLE policy_info
(
 policy_id    CHAR(4) NOT NULL,
 coverage     INT NOT NULL,
 sys_start    TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN IMPLICITLY HIDDEN,
 sys_end      TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END IMPLICITLY HIDDEN,
 ts_id        TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID IMPLICITLY HIDDEN,
 PERIOD SYSTEM_TIME (sys_start, sys_end)
) in policy_space;
```

Creating the `hist_policy_info` history table using the LIKE clause of the CREATE TABLE statement results in the history table inheriting the implicitly hidden attribute from the `policy_info` table. If you do not use the LIKE clause when creating the history table, then any columns marked as hidden in the system-period temporal table must also be marked as hidden in the associated history table.

Changing an existing table into a system-period temporal table

The following example adds timestamp columns and a SYSTEM_TIME period to an existing table (`employees`) enabling system-period temporal table functions.

```
ALTER TABLE employees
   ADD COLUMN sys_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN;
ALTER TABLE employees
   ADD COLUMN sys_end TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END;
ALTER TABLE employees
   ADD COLUMN ts_id TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID;
ALTER TABLE employees ADD PERIOD SYSTEM_TIME(sys_start, sys_end);
```

These new columns can be hidden by including the IMPLICITLY HIDDEN clause in the ALTER TABLE statement

A history table must be created and versioning added to finish this task.

Inserting data into a system-period temporal table

For a user, inserting data into a system-period temporal table is similar to inserting data into a regular table.

About this task

When inserting data into a system-period temporal table, the database manager automatically generates the values for the row-begin and row-end timestamp columns. The database manager also generates the transaction start-ID value that uniquely identifies the transaction that is inserting the row.

Procedure

To insert data into a system-period temporal table, use the INSERT statement to add data to the table. For example, the following data was inserted on January 31, 2010 (2010-01-31) to the table created in the example in “ Creating a system-period temporal table.”

```
INSERT INTO policy_info(policy_id, coverage)
   VALUES('A123',12000);

INSERT INTO policy_info(policy_id, coverage)
   VALUES('B345',18000);

INSERT INTO policy_info(policy_id, coverage)
   VALUES('C567',20000);
```

Results

The `policy_info` table now contains the following insurance coverage data. The `sys_start`, `sys_end`, and `ts_id` column entries were generated by the database manager.

Table 24. Data added to a system-period temporal table (policy_info)

policy_id	coverage	sys_start	sys_end	ts_id
A123	12000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
B345	18000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
C567	20000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000

The `his_policy_info` history table remains empty because no history rows are generated by an insert.

Table 25. History table (hist_policy_info) after insert

policy_id	coverage	sys_start	sys_end	ts_id

Note: The row-begin column, `sys_start`, represents the time when the row data became current. The database manager generates this value by using a reading of the system clock at the moment it executes the first data change statement in the transaction that generates the row. The database manager also generates the transaction start-ID column, `ts_id`, which captures the time when execution started for a transaction that impacts the row. In many cases the timestamp values for both these columns are the same because they result from the execution of the same transaction.

When multiple transactions are updating the same row, timestamp conflicts can occur. The database manager can resolve these conflicts by making adjustments to row-begin column timestamp values. In such cases, the values in row-begin column and transaction start-ID column would differ. The **Example** section in " Updating a system-period temporal table" provides more details on timestamp adjustments.

Updating data in a system-period temporal table

Updating data in a system-period temporal table results in rows that are added to its associated history table.

About this task

In addition to updating the values of specified columns in rows of the system-period temporal table, the UPDATE statement inserts a copy of the existing row into the associated history table. The history row is generated as part of the same transaction that updates the row. If a single transactions make multiple updates to the same row, only one history row is generated and that row reflects the state of the record before any changes were made by the transaction.

Note: Timestamp value conflicts can occur when multiple transactions are updating the same row. When these conflicts occur, the setting for the "systime_period_adj" database configuration parameter determines whether timestamp adjustments are made or if transactions should fail. The **Multiple changes to a row by different transactions** example in the **More examples** section provides more details. Application programmers might consider using SQLCODE or SQLSTATE values to handle potential timestamp value adjustment-related return codes from SQL statements.

Procedure

To update data in a system-period temporal table, use the UPDATE statement. For example, it was discovered that were some errors in the insurance coverage levels for a customer and the following data was updated on February 28, 2011 (2011-02-28) in the example table that had data added in the " Inserting data into a system-period temporal table" topic.
The following table contains the original `policy_info` table data.

Table 26. Original data in the system-period temporal table (policy_info)

policy_id	coverage	sys_start	sys_end	ts_id
A123	12000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
B345	18000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
C567	20000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000

- The coverage for policy C567 should be 25000.

  ```
  UPDATE policy_info
     SET coverage = 25000
     WHERE policy_id = 'C567';
  ```

  The update to policy C567 affects the system-period temporal table and its history table, causing the following things to occur:

  1. The coverage value for the row with policy C567 is updated to 25000.
  2. In the system-period temporal table, the database manager updates the `sys_start` and `ts_id` values to the date of the update.
  3. The original row is moved to the history table. The database manager updates the `sys_end` value to the date of the update. This row can be interpreted as the valid coverage for policy C567 from 2010-01-31-22.31.33.495925000000 to 2011-02-28-09.10.12.649592000000.

Table 27. Updated data in the system-period temporal table (policy_info)

policy_id	coverage	sys_start	sys_end	ts_id
A123	12000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
B345	18000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
C567	25000	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

Table 28. History table (hist_policy_info) after update

policy_id	coverage	sys_start	sys_end	ts_id
C567	20000	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000

More examples

This section contains more examples of updating system-period temporal tables.

Time specifications

In the following example, a time period is specified as part of the table update. The following update is run after the update in the preceding **Procedure** section.

```
UPDATE (SELECT * FROM policy_info
   FOR SYSTEM_TIME AS OF '2010-01-31-22.31.33.495925000000')
   SET coverage = coverage + 1000;
```

This update returns an error because it implicitly attempts to update history rows. The SELECT explicitly queries the `policy_info` table and implicitly queries its associated history table (`hist_policy_info`). The C567 row in `hist_policy_info` would be returned by the SELECT, but rows in a history table that were accessed implicitly cannot be updated.

Multiple changes to a row by different transactions

In the following example, two transactions are executing SQL statements against the `policy_info` table at the same time. In this example, the timestamps are simplified to a placeholder instead of a sample system clock value. For example, instead of 2010-01-31-22.31.33.495925000000, the example uses T1. Higher numbered placeholders indicate later actions within the transaction. For example, T5 is later than T4.

When you insert or update multiple rows within a single SQL transaction, the values for the row-begin column are the same for all the impacted rows. That value comes from a reading of the system clock at the moment the first data change statement in the transaction is executed. For example, all times associated with transaction ABC will have a time of T1.

```
Transaction ABC                         Transaction XYZ
T1: INSERT INTO policy_info
         (policy_id, coverage)
         VALUES ('S777',7000);
                                        T2: INSERT INTO policy_info
                                                 (policy_id, coverage)
                                                 VALUES ('T888',8000);
                                        T3: COMMIT;
T4: UPDATE policy_info
      SET policy_id = 'X999'
      WHERE policy_id = 'T888';
T5: INSERT INTO policy_info
         (policy_id, coverage)
         VALUES ('Y555',9000);
T6: COMMIT;
```

After the inserts at **T1** and **T2**, the `policy_info` table would look like this and the history table would be empty (`hist_policy_info`). The value **max** in the `sys_end` column is populated with the maximum default value for the TIMESTAMP(12) data type.

Table 29. Different transaction inserts to the policy_info table

policy_id	coverage	sys_start	sys_end	ts_id
S777	7000	T1	max	T1
T888	8000	T2	max	T2

After the update by transaction ABC at time T4, the policy information looks like the following tables. All the rows in the `policy_info` table reflect the insert and update activities from transaction ABC. The `sys_start` and `ts_id` columns for these rows are populated with time T1, which is the time of the first data change statement in transaction ABC. The policy information inserted by transaction XYZ was updated and the original row is moved to the history table.

Table 30. Different transactions after update to the policy_info table

policy_id	coverage	sys_start	sys_end	ts_id
S777	7000	T1	max	T1
X999	8000	T1	max	T1

Table 31. History table after different transactions update (hist_policy_info)

policy_id	coverage	sys_start	sys_end	ts_id
T888	8000	T2	T1	T2

The history table shows a `sys_end` time that is less than the `sys_start`. In this situation, the update at time T4 could not execute and transaction ABC would fail (SQLSTATE 57062, SQLCODE SQL20528N). To avoid such failures, the **`systime_period_adj`** database configuration parameter can be set to *YES* which allows the database manager to adjust the row-begin timestamp (SQLSTATE 01695, SQLCODE SQL5191W). The `sys_start` timestamp for the time T4 update in transaction ABC is set to time T2 plus a delta (T2+delta). This adjustment only applies to the time T4 update, all other changes made by transaction ABC would continue to use the time T1 timestamp (for example, the insert of the policy with `policy_id` Y555). After this adjustment and the completion of transaction ABC, the insurance policy tables would contain the following data:

Table 32. Different transactions after time adjustment (policy_info)

policy_id	coverage	sys_start	sys_end	ts_id
S777	7000	T1	max	T1
X999	8000	T2+delta	max	T1
Y555	9000	T1	max	T1

Table 33. History table after time adjustment (hist_policy_info)

policy_id	coverage	sys_start	sys_end	ts_id
T888	8000	T2	T2+delta	T2

Multiple changes to a row in the same transaction

In the following example, a transaction makes multiple changes to a row. Using the insurance policy tables from the previous example, transaction ABC continues and updates the policy with `policy_id` X999 at time T6 (originally T6 was a COMMIT statement).

Transaction ABC

```
T6: UPDATE policy_info SET policy_id = 'R111' WHERE policy_id = 'X999';
T7: COMMIT;
```

This row has now experienced the following changes:

1. Created as policy T888 by transaction XYZ at time T2.
2. Updated to policy X999 by transaction ABC at time T4.
3. Updated to policy R111 by transaction ABC at time T6.

When a transaction makes multiple updates to the same row, the database manager generates a history row only for the first change. This, results in the following tables:

Table 34. Same transaction after updates (policy_info)

policy_id	coverage	sys_start	sys_end	ts_id
S777	7000	T1	max	T1
R111	8000	T1	max	T1
Y555	9000	T1	max	T1

Table 35. History table after same transaction update (hist_policy_info)

policy_id	coverage	sys_start	sys_end	ts_id
T888	8000	T2	T2+delta	T2

The database manager uses the transaction-start-ID (`ts_id`) to uniquely identify the transaction that changes the row. When multiple rows are inserted or updated within a single SQL transaction, then the values for the transaction start-ID column are the same for all the rows and are unique from the values generated for this column by other transactions. Before generating a history row, the database manager determines that the last update to the row was for the transaction that started at time T1 (`ts_id` is T1), which is the same transaction start time for the transaction that makes the current change and so no history row is generated. The `sys_start` value for the row in the `policy_info` table is changed to time T1.

Updating a view

A view that references a system-period temporal table or a bitemporal table can be updated only if the view definition does not contain a FOR SYSTEM_TIME clause. The following UPDATE statement updates the `policy_info` table and generates history rows.

```
CREATE VIEW viewA AS SELECT * FROM policy_info;
UPDATE viewA SET col2 = col2 + 1000;
```

A view that references a system-period temporal table or a bitemporal table with a view definition containing a FOR SYSTEM_TIME clause can be made updatable by defining an INSTEAD OF trigger. The following example updates the `regular_table` table.

```
CREATE VIEW viewB AS SELECT * FROM policy_info;
   FOR SYSTEM_TIME BETWEEN
   TIMESTAMP '2010-01-01 10:00:00' AND TIMESTAMP '2011-01-01 10:00:00';

CREATE TRIGGER update INSTEAD OF UPDATE ON viewB
   REFERENCING NEW AS n FOR EACH ROW
   UPDATE regular_table SET col1 = n.id;

UPDATE viewB set id = 500;
```

Deleting data from a system-period temporal table

Deleting data from a system-period temporal table removes rows from the table and adds rows to the associated history table. The rows are added with the appropriate system timestamps.

About this task

In addition to deleting the specified rows of the system-period temporal table, the DELETE FROM statement moves a copy of the existing row into the associated history table before the row is deleted from the system-period temporal table.

Procedure

To delete data from a system-period temporal table, use the DELETE FROM statement. For example, the owner of policy B345 decides to cancel insurance coverage. The data was deleted on September 1, 2011 (2011-09-01) from the table that was updated in the " Updating data in a bitemporal table" topic.

```
DELETE FROM policy_info WHERE policy_id = 'B345';
```

Results

The original `policy_info` table data is as follows:

Table 36. Data in the system-period temporal table (policy_info) before the DELETE statement

policy_id	coverage	sys_start	sys_end	ts_id
A123	12000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
B345	18000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
C567	25000	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

The deletion of policy B345 affects the system-period temporal table and its history table, causing the following things to occur:

1. The row where the `policy_id` column value is B345 is deleted from the system-period temporal table.
2. The original row is moved to the history table. The database manager updates the `sys_end` column value to the date of the delete.

Table 37. Data in the system-period temporal table (policy_info) after the DELETE statement

policy_id	coverage	sys_start	sys_end	ts_id
A123	12000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
C567	25000	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

Table 38. History table (hist_policy_info) after delete

policy_id	coverage	sys_start	sys_end	ts_id
C567	20000	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000

Table 38. History table (hist_policy_info) after delete (continued)

policy_id	coverage	sys_start	sys_end	ts_id
B345	18000	2010-01-31-22.31.33.495925000000	2011-09-01-12.18.22.959254000000	2010-01-31-22.31.33.495925000000

Example

This section contains more examples of delete operations on system-period temporal tables.

Time specifications

In the following example, a time period is specified as part of the DELETE statement. The following delete is run after the delete in the preceding **Procedure** section.

```
DELETE FROM (SELECT * FROM policy_info
   FOR SYSTEM_TIME AS OF '2010-01-31-22.31.33.495925000000')
   WHERE policy_id = C567;
```

This DELETE statement returns an error. The SELECT statement explicitly queries the `policy_info` table and implicitly queries its associated history table (`hist_policy_info`). The row with a `policy_id` column value of C567 in the `hist_policy_info` table would be returned by the SELECT statement, but rows in a history table that were accessed implicitly cannot be deleted.

Querying system-period temporal data

Querying a system-period temporal table can return results for a specified point or period in time. Those results can include current values and previous historic values.

About this task

When querying a system-period temporal table, you can include FOR SYSTEM_TIME in the FROM clause. Using FOR SYSTEM_TIME specifications, you can query the current and past state of your data. Time periods are specified as follows:

AS OF *value1*

Includes all the rows where the begin value for the period is less than or equal to *value1* and the end value for the period is greater than *value1*. This enables you to query your data as of a certain point in time.

FROM *value1* **TO** *value2*

Includes all the rows where the value of the begin column for the specified period in the row is less than *value2*, and the value of the end column for the specified period in the row is greater than *value1*. No rows are returned if *value1* is greater than or equal to *value2*.

BETWEEN *value1* **AND** *value2*

Includes all the rows where any time period overlaps any point in time between *value1* and *value2*. A row is returned if the begin value for the period is less than or equal to *value2* and the end value for the period is greater than *value1*.

See the following section for some sample queries.

Procedure

To query a system-period temporal table, use the SELECT statement. For example, each of the following queries requests policy information from the result tables in the Deleting data from a system-period temporal table topic. Each query uses a variation of the FOR SYSTEM_TIME specification.
The `policy_info` table and its associated history table are as follows:

Table 39. System-period temporal table: policy_info

policy_id	coverage	sys_start	sys_end	ts_id
A123	12000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
C567	25000	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

Table 40. History table: hist_policy_info

policy_id	coverage	sys_start	sys_end	ts_id
C567	20000	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
B345	18000	2010-01-31-22.31.33.495925000000	2011-09-01-12.18.22.959254000000	2010-01-31-22.31.33.495925000000

- Query with no time period specification. For example:

  ```
  SELECT policy_id, coverage
     FROM policy_info
     where policy_id = 'C567'
  ```

  This query returns one row. The SELECT queries only the `policy_info` table. The history table is not queried because FOR SYSTEM_TIME was not specified.

  ```
  C567, 25000
  ```

- Query with FOR SYSTEM_TIME AS OF specified. For example:

  ```
  SELECT policy_id, coverage
     FROM policy_info
     FOR SYSTEM_TIME AS OF
        '2011-02-28-09.10.12.649592000000'
  ```

  This query returns three rows. The SELECT queries both the `policy_info` and the `hist_policy_info` tables. The begin column of a period is inclusive, while the end column is exclusive. The history table row with a sys_end column value of 2011-02-28-22.31.33.495925000000 equals *value1*, but it must be less than *value1* in order to be returned.

  ```
  A123, 12000
  C567, 25000
  B345, 18000
  ```

- Query with FOR SYSTEM_TIME FROM..TO specified. For example:

  ```
  SELECT policy_id, coverage, sys_start, sys_end
     FROM policy_info
     FOR SYSTEM_TIME FROM
        '0001-01-01-00.00.00.000000' TO '9999-12-30-00.00.00.000000000000'
     where policy_id = 'C567'
  ```

This query returns two rows. The SELECT queries both the `policy_info` and the `hist_policy_info` tables.

```
C567, 25000, 2011-02-28-09.10.12.649592000000, 9999-12-30-00.00.00.000000000000
C567, 20000, 2010-01-31-22.31.33.495925000000, 2011-02-28-09.10.12.649592000000
```

- Query with FOR SYSTEM_TIME BETWEEN..AND specified. For example:

```
SELECT policy_id, coverage
   FROM policy_info
   FOR SYSTEM_TIME BETWEEN
      '2011-02-28-09.10.12.649592000000' AND '9999-12-30-00.00.00.000000000000'
```

This query returns three rows. The SELECT queries both the `policy_info` and the `hist_policy_info` tables. The rows with a sys_start column value of 2011-02-28-09.10.12.649592000000 are equal to *value1* and are returned because the begin time of a period is included. The rows with a sys_end column value of 2011-02-28-09.10.12.649592000000 are equal to *value1* and are not returned because the end time of a period is not included.

```
A123, 12000
C567, 25000
B345, 18000
```

More examples

This section contains more querying system-period temporal table examples.

Query using other valid date or timestamp values

The `policy_info` table was created with its time-related columns declared as `TIMESTAMP(12)`, so queries using any other valid date or timestamp value are converted to use TIMESTAMP(12) before execution. For example:

```
SELECT policy_id, coverage
   FROM policy_info
   FOR SYSTEM_TIME AS OF '2011-02-28'
```

is converted and executed as:

```
SELECT policy_id, coverage
   FROM policy_info
   FOR SYSTEM_TIME AS OF '2011-02-28-00.00.00.000000000000'
```

Querying a view

A view can be queried as if it were a system-period temporal table. FOR SYSTEM_TIME specifications can be specified after the view reference.

```
CREATE VIEW policy_2011(policy, start_date)
   AS SELECT policy_id, sys_start FROM policy_info;

SELECT * FROM policy_2011 FOR SYSTEM_TIME BETWEEN
   '2011-01-01-00.00.00.000000' AND '2011-12-31-23.59.59.999999999999';
```

The SELECT on the view `policy_2011` queries both the `policy_info` and the `hist_policy_info` tables. Returned are all policies that were active at anytime in 2011 and includes the date the policies were started.

```
A123, 2010-01-31-22.31.33.495925000000
C567, 2011-02-28-09.10.12.649592000000
C567, 2010-01-31-22.31.33.495925000000
B345, 2010-01-31-22.31.33.495925000000
```

If a view definition contains a period specification, then queries against the view cannot contain period specifications. The following statements return an error due to multiple period specifications:

```
CREATE VIEW all_policies AS SELECT * FROM policy_info;
   FOR SYSTEM_TIME AS OF '2011-02-28-09.10.12.649592000000';

SELECT * FROM all_policies FOR SYSTEM_TIME BETWEEN
   '2011-01-01-00.00.00.000000' AND '2011-12-31-23.59.59.999999999999';
```

Setting the system time for a session

Setting the system time with the CURRENT TEMPORAL SYSTEM_TIME special register can reduce or eliminate the changes required when running an application against different points in time.

About this task

When you have an application that you want to run against a system-period temporal table to query the state of your business for a number of different dates, you can set the date in a special register. If you need to query your data as of today, as of the end of the last quarter, and as of the same date from last year, it might not be possible to change the application and add AS OF specifications to each SQL statement. This restriction is likely the case when you are using packaged applications. To address such scenarios, you can use the CURRENT TEMPORAL SYSTEM_TIME special register to set the date or timestamp at the session level.

Setting the CURRENT TEMPORAL SYSTEM_TIME special register does not affect regular tables. Only queries on temporal tables with versioning enabled (system-period temporal tables and bitemporal tables) use the time set in the special register. There is also no affect on DDL statements. The special register does not apply to any scans run for referential integrity processing. .

Important: When the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, data modification statements like INSERT, UPDATE, and DELETE against system-period temporal tables are blocked. If the special register was set to some time in the past, for example five years ago, then allowing data modification operations might result in changes to your historical data records. Utilities like IMPORT and LOAD are also blocked against system-period temporal tables when the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value.

The BIND command contains the SYSTIMESENSITIVE option that indicates whether references to system-period temporal tables in static and dynamic SQL statements are affected by the value of the CURRENT TEMPORAL SYSTEM_TIME special register. For SQL procedures, use the SET_ROUTINE_OPTS procedure to set the bind-like options, called query compiler variables.

Procedure

When this special register is set to a non-null value, applications that issue a query will return data as of that date or timestamp. The following examples request information from the result tables in the Deleting data from a system-period temporal table topic.

- Set the special register to the current timestamp and query data from one year ago. Assuming a current timestamp of 2011-05-17-14.45.31.434235000000:
  ```
  SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT TIMESTAMP - 1 YEAR;
  SELECT policy_id, coverage FROM policy_info;
  ```
- Set the special register to a timestamp and reference a system-period temporal table in view definitions.

```
CREATE VIEW view1 AS SELECT policy_id, coverage FROM policy_info;
CREATE VIEW view2 AS SELECT * FROM regular_table
   WHERE col1 IN (SELECT coverage FROM policy_info);
SET CURRENT TEMPORAL SYSTEM_TIME = TIMESTAMP '2011-02-28-09.10.12.649592000000';
SELECT * FROM view1;
SELECT * FROM view2;
```

- Set the special register to the current timestamp and issue a query that contains a time period specification. Assuming a current timestamp of 2011-05-17-14.45.31.434235000000:

```
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT TIMESTAMP - 1 YEAR;
SELECT *
  FROM policy_info FOR SYSTEM_TIME AS OF '2011-02-28-09.10.12.649592000000';
```

Results

The policy_info table and its associated history table are as follows:

Table 41. Data in the system-period temporal table (policy_info) after the DELETE statement

policy_id	coverage	sys_start	sys_end	ts_id
A123	12000	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
C567	25000	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

Table 42. History table (hist_policy_info) after delete

policy_id	coverage	sys_start	sys_end	ts_id
C567	20000	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
B345	18000	2010-01-31-22.31.33.495925000000	2011-09-01-12.18.22.959254000000	2010-01-31-22.31.33.495925000000

- The request for data from one year ago queries the policy_info table as of 2010-05-17-14.45.31.434235000000. The query is implicitly rewritten to:

  ```
  SELECT policy_id, coverage FROM policy_info
     FOR SYSTEM_TIME AS OF TIMESTAMP '2010-05-17-14.45.31.434235000000';
  ```

  The SELECT queries both the policy_info and the hist_policy_info tables and returns:

  ```
  A123, 12000
  C567, 20000
  B345, 18000
  ```

- The query on view1 is implicitly rewritten to:

  ```
  SELECT * FROM view1 FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME;
  ```

  The query is then rewritten to:

  ```
  SELECT policy_id, coverage FROM policy_info
     FOR SYSTEM_TIME AS OF TIMESTAMP '2011-02-28-09.10.12.649592000000';
  ```

  The SELECT queries both the policy_info and the hist_policy_info tables and returns:

  ```
  A123, 12000
  C567, 25000
  B345, 18000
  ```

The query on view2 involves a view on a regular table that references a system-period temporal table, causing an implicit relationship between a regular table and the special register. The query is implicitly rewritten to:

```
SELECT * FROM view2 FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME;
```

The query is then rewritten to:

```
SELECT * FROM regular_table WHERE col1 in (SELECT coverage FROM policy_info
   FOR SYSTEM_TIME AS OF TIMESTAMP '2011-02-28-09.10.12.649592000000');
```

The SELECT returns rows where col1 values match values in coverage.

- An error is returned because there are multiple time period specifications. The special register was set to a non-null value and the query also specified a time.

Dropping a system-period temporal table

Dropping a system-period temporal table also drops its associated history table and any indexes defined on the history table.

Before you begin

To drop a system-period temporal table, you must be authorized to drop its history table.

About this task

A history table is implicitly dropped when its associated system-period temporal table is dropped. A history table cannot be explicitly dropped by using the DROP statement.

To avoid losing historical data when a system-period temporal table is dropped, you can either create the history table with the RESTRICT ON DROP attribute or alter the history table by adding the RESTRICT ON DROP attribute. If you try to drop a system-period temporal table and its history table has the RESTRICT ON DROP attribute, the drop of the system-period temporal table fails (SQLSTATE 42893). In such cases, you must break the link between the system-period temporal table and the history table by removing the VERSIONING attribute and then rerun the DROP statement.

When a table is altered to drop VERSIONING, all packages with the versioning dependency on the table are invalidated. Other dependent objects, for example, views or triggers are marked invalid 'N' in the system catalog. Auto-revalidation is done. Any objects failing revalidation are left as invalid in the catalog. Some objects can become valid after only explicit user action.

Procedure

To drop a system-period temporal table and its associated history table:

1. Optional: Protect historical data from deletion:
   a. If the history table was not created with the RESTRICT ON DROP attribute, alter the history table to set the RESTRICT ON DROP attribute. For example, if audit requirements made it necessary to preserve the history of insurance policies then the history table must be protected.

      ```
      ALTER TABLE hist_policy_info ADD RESTRICT ON DROP;
      ```

   b. Break the link between the system-period temporal table and a history table with RESTRICT ON DROP attribute by removing the VERSIONING attribute. For example:

      ```
      ALTER TABLE policy_info DROP VERSIONING;
      ```

2. Drop the system-period temporal table with the DROP statement. For example, the insurance policy tables created in the example in the Creating a system-period temporal table topic are no longer required.

   ```
   DROP TABLE policy_info;
   ```

Results

The preceding commands affect the policy_info and hist_policy_info tables as follows:

- The DROP statement explicitly drops the system-period temporal table and implicitly drops the associated history table. The policy_info and hist_policy_info tables are deleted. Any objects that are directly or indirectly dependent on those tables are either deleted or made inoperative.
- After the RESTRICT ON DROP attribute is associated with the history table, any attempt to drop the policy_info table would fail (SQLSTATE 42893). A system-period temporal table can also be created or altered to use the RESTRICT ON DROP attribute.
- After the link between the system-period temporal table and its history table is broken, the policy_info table can be dropped and the hist_policy_info history table would remain.

Dropping table spaces

If a table space contains a history table, but does not contain the associated system-period temporal table, that table space cannot be explicitly dropped. For example, using the insurance policy tables that were created in the policy_space and hist_space table spaces, the following statement is blocked:

```
DROP TABLESPACE hist_space;
```

If table space that contains a history table and the table space containing the associated system-period temporal table are included together, then the statement is allowed. For example, the following statement would succeed:

```
DROP TABLESPACE policy_space hist_space;
```

A history table is implicitly dropped when the table space for its associated system-period temporal table is dropped. For example, the following statement would drop the hist_policy_info history table:

```
DROP TABLESPACE policy_space;
```

Utilities and tools

There are a number of tools and utilities available for you to work with and manage the data in your temporal tables.

The following tools are available to work with and manage temporal tables:

- Import data (see “Import” on page 267)
- Load data (see “Load” on page 267)
- Online table move (see “ADMIN_MOVE_TABLE procedure” on page 268)
- Quiesce table space (see “QUIESCE TABLESPACES FOR TABLE command” on page 268)

- Replication (see “Replication” on page 268)
- Roll forward (see “Roll forward” on page 269)
- ADMIN_COPY_SCHEMA procedure (see ADMIN_COPY_SCHEMA)

Import

When importing data into system-period temporal tables, you use file type modifiers to ignore any content in the external file that might be applied to the database manager generated columns in the system-period temporal table. The following modifiers are available when importing data into a system-period temporal table.

periodignore
: Use this modifier to inform the import utility that data for the SYSTEM_TIME period columns is present in the external file, but should be ignored. When this modifier is specified, all time period column values are generated by the utility.

periodmissing
: Use this modifier to advise the import utility that the external data file does not contain any data for the SYSTEM_TIME period columns. When this modifier is specified, all time period column values are generated by the utility.

transactionidignore
: Use this modifier to inform the import utility that data for the transaction start-ID column is present in the external file, but should be ignored. When this modifier is specified, the value for the transaction start-ID column is generated by the utility.

transactionidmissing
: Use this modifier to advise the import utility that the external data file does not contain any data for the transaction start-ID column. When this modifier is specified, the value for the transaction start-ID column is generated by the utility.

Unlike the load utility, the import utility does not have modifiers that override the GENERATED ALWAYS columns.

Load

When loading data into system-period temporal tables, you use file type modifiers to either ignore any data in the external file that might be applied to the database manager generated columns, or to load user-supplied values to those generated columns. The following modifiers are available when loading data into a system-period temporal table. LOAD REPLACE is blocked on system-period temporal tables.

periodignore
: Use this modifier to inform the load utility that data for the SYSTEM_TIME period columns is present in the external file, but should be ignored. When this modifier is specified, all time period column values are generated by the utility.

periodmissing
: Use this modifier to advise the load utility that the external data file does

not contain any data for the SYSTEM_TIME period columns. When this modifier is specified, all time period column values are generated by the utility.

periodoverride
Use this modifier to instruct the load utility to accept user-supplied values for the SYSTEM_TIME period row-begin and row-end columns. This modifier overrides the GENERATED ALWAYS clause. This modifier can be useful when you want to maintain history data and load data that includes time stamps into a system-period temporal table. When this modifier is used, any rows with no data or NULL data in the row-begin and row-end columns are rejected.

transactionidignore
Use this modifier to inform the load utility that data for the transaction start-ID column is present in the external file, but should be ignored. When this modifier is specified, the value for the transaction start-ID column is generated by the utility.

transactionidmissing
Use this modifier to advise the load utility that the external data file does not contain any data for the transaction start-ID column. When this modifier is specified, the value for the transaction start-ID column is generated by the utility.

transactionidoverride
Use this modifier to instruct the load utility to accept user-supplied values for the transaction start-ID column. This modifier overrides the GENERATED ALWAYS clause. When this modifier is used, any rows with no data or NULL data in a transaction start-ID column are rejected.

ADMIN_MOVE_TABLE procedure

When using the ADMIN_MOVE_TABLE stored procedure to move data in an active system-period temporal table into a new table with the same name, the following actions are blocked.

- Alter table operations that change the definition of the system-period temporal table or the associated history table are blocked during online move operations.
- The KEEP option of ADMIN_MOVE_TABLE is unavailable for system-period temporal tables

The online-table-move operation is not supported for history tables.

QUIESCE TABLESPACES FOR TABLE command

When running the QUIESCE TABLESPACES FOR TABLE command on a system-period temporal table, all the table spaces associated with the system-period temporal table and its history table are quiesced. When running the command against a history table, all the table spaces associated with the history table and the associated system-period temporal table are quiesced.

Replication

When replicating a system-period temporal table, columns with following generated attributes cannot participate in the replication if the target is another system-period temporal table:

- GENERATED ALWAYS AS ROW BEGIN

- GENERATED ALWAYS AS ROW END
- GENERATED ALWAYS AS TRANSACTION START ID

Similarly, when replicating a bitemporal table, columns with following generated attributes cannot participate in the replication if the target is another bitemporal table:

Roll forward

When the table space for a system-period temporal table or a bitemporal table is rolled-forward to a point in time, the table space for the associated history table also must be rolled-forward to the same point in time in the same ROLLFORWARD statement. Similarly when the table space for a history table is rolled-forward to a point in time, the table space for the system-period temporal table or a bitemporal table also must be rolled-forward to the same point in time. You can, however, recover the table space for the system-period temporal table or the table space for the history table to end of logs individually.

ADMIN_COPY_SCHEMA procedure

The ADMIN_COPY_SCHEMA procedure is used to copy a specific schema and all objects contained in it. The new target schema objects are created using the same object names as the objects in the source schema, but with the target schema qualifier. The ADMIN_COPY_SCHEMA procedure is supported for system-period temporal tables. The procedure requires that both system-period temporal table and the history table are in the same schema, otherwise neither table is copied and an error is recorded.

Schema changes

In order to maintain the integrity of the relationship between the system-period temporal table and its associated history table, only certain changes to the schema of a system-period temporal table are permitted. Any changes that would result in the loss of data are restricted.

You can make the following changes to your system-period temporal tables. These changes are implicitly propagated to the associated history table if you have the appropriate privileges. These changes cannot be explicitly made to the history table.

- ADD COLUMN (except generated columns)
- RENAME COLUMN
- ALTER COLUMN (in cases where no history data is lost). For example, altering the data type of a column from VARCHAR(50) to VARCHAR(100), or from INTEGER to DECIMAL is permitted. However, the reverse change from VARCHAR(100) to VARCHAR(50), or from DECIMAL to INTEGER is blocked because it would reduce the length or precision of a column and likely cause a loss of data.

You cannot make the following changes to your system-period temporal tables because data would likely be lost:

- DROP COLUMN
- ADD COLUMN (generated)
- ALTER COLUMN (in cases where history data might be lost). For example, altering the data type of a column from VARCHAR(50) to VARCHAR(100), or from INTEGER to DECIMAL is permitted. However, the reverse change from

VARCHAR(100) to VARCHAR(50), or from DECIMAL to INTEGER is blocked because it would reduce the length or precision of a column.

Versioning establishes a link between your system-period temporal table and its associated history table. When versioning is active, certain ALTER TABLE operations are blocked on system-period temporal tables and history tables.

- ALTER TABLE DROP PERIOD
- ALTER TABLE ADD MATERIALIZED
- ALTER TABLE ACTIVATE NOT LOGGED INITIALLY
- ALTER TABLE ADD SECURITY POLICY
- ALTER TABLE DROP SECURITY POLICY
- ALTER TABLE SECURED WITH ALTER

ALTER TABLE operations that are not shown in the previous list are supported, but are not propagated from a system-period temporal table to its history table.

Additionally, RENAME INDEX and RENAME TABLE are supported, but are not propagated from a system-period temporal table to its history table.

Cursors and system-period temporal tables

Cursors used to update or delete rows for a query that potentially references rows in a history table must be read-only.

In the following example, the statement runs successfully because the cursor `appcur` is read-only.

```
DECLARE appcur CURSOR FOR SELECT * FROM policy_info
   FOR SYSTEM_TIME AS OF '2011-02-28';
```

However, the following statement results in an error because any cursor references to the history rows are not read-only:

```
DECLARE appcur CURSOR FOR SELECT * FROM policy_info
   FOR SYSTEM_TIME AS OF '2011-02-28' FOR UPDATE;
```

Table partitioning and system-period temporal tables

A system-period temporal table can have its table data divided across multiple storage objects called data partitions. A history table associated with a system-period temporal table can also be partitioned.

When versioning is enabled, the following behaviors apply when attaching a partition to a system-period temporal table or detaching a partition from a system-period temporal table:

Attaching partitions

- A table can be attached to a system-period temporal table while versioning is enabled.
- The table being attached must contain all three timestamp columns (ROW BEGIN, ROW END, and TRANSACTION START ID). These timestamp columns must have the same definitions as those columns in the system-period temporal table.
- The table being attached does not require a SYSTEM_TIME period definition.
- While versioning is enabled, the SET INTEGRITY ... FOR EXCEPTION statement cannot be run because moving exception rows into an

exception table would result in lost history. Because the exception rows are not recorded in the history table, the auditability of the data in your system-period temporal table and its associated history table is jeopardized. You can temporarily stop versioning, run the SET INTEGRITY ... FOR EXCEPTION statement, and then enable versioning again.

Detaching partitions

- A table cannot be detached from a system-period temporal table while versioning is enabled. You can stop versioning and then detach a partition from the base table. The detached partition becomes an independent table. Detaching a partition from a history table does not require that you stop versioning.
- A detached partition retains all three timestamp columns (ROW BEGIN, ROW END, and TRANSACTION START ID), but not the SYSTEM_TIME period definition.
- The rows in a detached partition are not automatically moved to the history table. If you want to maintain the history, then the rows must be moved manually. If you manually move the rows to the history table, you should change the ROW END timestamp to the point-in-time when the rows changed from being current to being history. Without these changes, time-related queries might return unexpected results.

Data access control for system-period temporal tables

Row and column access control can be defined on both a system-period temporal table and its associated history table.

Row and column access control (RCAC) is a layer of data security that controls access to a table at the row level, column level, or both. RCAC can be applied to system-period temporal tables and history tables. When RCAC is only activated for a system-period temporal table, the database manager automatically activates row access control on the history table and creates a default row permission for the history table.

When the history table is protected by the default row permission, updates and deletes still generate history rows in the history table. When an AS OF query is issued against a system-period temporal table, the RCAC row permissions and column masks for the system-period temporal table are also applied to the rows returned from the history table.

If a history table is accessed directly, then any row and column rules defined on the history table are applied.

Restrictions for system-period temporal tables

System-period temporal tables are subject to a number of restrictions. These restrictions can impact your implementation of system-period temporal tables.

Following are the restrictions for system-period temporal tables:

- Label-based access control (LBAC) is not supported on system-period temporal tables. While system-period data versioning is enabled, adding row and column labels to either a system-period temporal table or a history table is blocked. When versioning is enabled with an ALTER TABLE statement, the database manager ensures that both the system-period temporal table and the history table do not have rows or columns secured with labels.

- ALTER operations that cause a potential loss of data are not supported on system-period temporal tables.
- ALTER TABLE ACTIVATE NOT LOGGED INITIALLY statements are blocked for both the system-period temporal table and the history table.
- A system-period temporal table cannot be altered to become a materialized query table (MQT).
- Utilities that delete data from system-period temporal tables are blocked, including LOAD REPLACE and IMPORT REPLACE.
- The TRUNCATE statement is not supported against a system-period temporal table.
- The following schema-changing operations are not supported against system-period temporal tables:
  - ALTER TABLE DROP COLUMN
  - ALTER TABLE ALTER COL (Altering string data types to a type that requires data truncation is not supported. Altering numeric data types to a lower precision type is also not supported).
  - ALTER TABLE ADD GENERATED COLUMN
- For point-in-time recovery, if a table space that contains the system-period temporal table is being rolled forward to a point in time, the table space that contains the associated history table must also be rolled forward to the same point in time as a set. Similarly when the table space for a history table is rolled-forward to a point in time, the table space for the system-period temporal table or a bitemporal table also must be rolled-forward to the same point in time. However, rolling only the system-period temporal tables table space (or the history tables table space) to the end of logs is allowed.
- A nickname can be created over a remote system-period temporal table, but the temporal information is not exposed and temporal operations over nicknames are not supported. For instance, temporal data definition operations and temporal queries against federated nicknames are blocked.
- IMPORT and LOAD operations into system-period temporal tables are blocked if the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value.

Application-period temporal tables

An application-period temporal table is a table that stores the in effect aspect of application data. Use an application-period temporal table to manage data based on time criteria by defining the time periods when data is valid.

Similar to a system-period temporal table, an application-period temporal table includes a BUSINESS_TIME period with columns that indicate the time period when the data in that row is valid or in effect. You provide the begin time and end time for the BUSINESS_TIME period associated with each row. However, unlike a system time-period temporal table, there is no separate history table. Past, present, and future effective dates and their associated business data are maintained in a single table. You can control data values by BUSINESS_TIME period and use application-period temporal tables for modeling data in the past, present, and future.

BUSINESS_TIME period

The BUSINESS_TIME period columns for an application-period temporal table record when the version of a row is valid from a user or business application perspective.

The BUSINESS_TIME period contains a pair of DATE or TIMESTAMP(*p*) columns where *p* can be from 0 to 12. These columns are populated by you or a business application. The two columns in a BUSINESS_TIME period denote the start and end of the validity period. These columns differs from SYSTEM_TIME period columns where the time period values are generated by the database manager. The BUSINESS_TIME period columns must be defined as NOT NULL and must not be generated columns.

A BUSINESS_TIME period is inclusive-exclusive. The start of the validity period is included in the BUSINESS_TIME, while the end is excluded.

Whenever a BUSINESS_TIME period is defined on a table, an implicit check constraint named DB2_GENERATED_CHECK_CONSTRAINT_FOR_BUSINESS_TIME is generated to ensure that the value for the end of the validity period is greater than the value for the start of the validity period. If a constraint with the same name exists, then an error is returned. This constraint is useful when supporting operations that explicitly input data into these columns, such as an insert or load operation.

An application-period temporal table can be defined so that rows with the same key do not have any overlapping periods of BUSINESS_TIME. For example, this restriction would prevent two versions of an insurance policy from being in effect at any point in time. These controls can be achieved by adding a BUSINESS_TIME WITHOUT OVERLAPS clause to a primary key, unique constraint specification, or create unique index statement. The enforcement is handled by the database manager. This control is optional.

Creating an application-period temporal table

Creating an application-period temporal table results in a table that manages data based on when its data is valid or in effect.

About this task

When creating an application-period temporal table, include a BUSINESS_TIME period that indicates when the data in a row is valid. You can optionally define that overlapping periods of BUSINESS_TIME are not allowed and that values are unique with respect to any period. The example in the following section shows the creation of a table that stores policy information for the customers of an insurance company.

Procedure

To create an application-period temporal table:

1. Create a table with a BUSINESS_TIME period. For example:

```
CREATE TABLE policy_info
(
 policy_id     CHAR(4) NOT NULL,
 coverage      INT NOT NULL,
 bus_start     DATE NOT NULL,
 bus_end       DATE NOT NULL,
 PERIOD BUSINESS_TIME (bus_start, bus_end)
 );
```

2. Optional: Create a unique index that prevents overlapping periods of BUSINESS_TIME for the same `policy_id`. For example:

```
CREATE UNIQUE INDEX ix_policy
   ON policy_info (policy_id, BUSINESS_TIME WITHOUT OVERLAPS);
```

Results

The policy_info table stores the insurance coverage level for a customer. The BUSINESS_TIME period-related columns (bus_start and bus_end) indicate when an insurance coverage level is valid.

Table 43. Created application-period temporal table (policy_info)

policy_id	coverage	bus_start	bus_end

The ix_policy index, with BUSINESS_TIME WITHOUT OVERLAPS as the final column in the index key column list, ensures that there are no overlapping time periods for customer insurance coverage levels.

Example

This section contains more examples of creating application-period temporal tables.

Changing an existing table into an application-period temporal table

The following example adds time columns and a BUSINESS_TIME period to an existing table (employees) enabling application-period temporal table functionality. Adding the BUSINESS_TIME WITHOUT OVERLAPS clause ensures that an employee is listed only once in any time period.

```
ALTER TABLE employees ADD COLUMN bus_start DATE NOT NULL;
ALTER TABLE employees ADD COLUMN bus_end DATE NOT NULL;
ALTER TABLE employees ADD PERIOD BUSINESS_TIME(bus_start, bus_end);
ALTER TABLE employees ADD CONSTRAINT uniq
   UNIQUE(employee_id, BUSINESS_TIME WITHOUT OVERLAPS);
```

Preventing overlapping periods of time

In the “Procedure” section, an index ensures that there are no overlapping BUSINESS_TIME periods. In the following alternative example, a PRIMARY KEY declaration is used when creating the policy_info table, ensuring that overlapping periods of BUSINESS_TIME are not allowed. This means that there cannot be two versions of the same policy that are valid at the same time.

```
CREATE TABLE policy_info
(
  policy_id     CHAR(4) NOT NULL,
  coverage      INT NOT NULL,
  bus_start     DATE NOT NULL,
  bus_end       DATE NOT NULL,
  PERIOD BUSINESS_TIME(bus_start, bus_end),
  PRIMARY KEY(policy_id, BUSINESS_TIME WITHOUT OVERLAPS)
);
```

Ensuring uniqueness for periods of time

The following example creates a product_availability table where a company tracks the products it distributes, the suppliers of those products, and the prices the suppliers charge. Multiple suppliers can provide the same product at the same time, but a PRIMARY KEY declaration ensures that a single supplier can only charge one price at any given point in time.

```
CREATE TABLE product_availability
(
  product_id     CHAR(4) NOT NULL,
  supplier_id    INT NOT NULL,
  product_price  DECIMAL NOT NULL
  bus_start      DATE NOT NULL,
  bus_end        DATE NOT NULL,
  PERIOD BUSINESS_TIME(bus_start, bus_end),
  PRIMARY KEY(product_id, supplier_id, BUSINESS_TIME WITHOUT OVERLAPS)
);
```

If the PRIMARY KEY was defined as

```
  PRIMARY KEY(product_id, BUSINESS_TIME WITHOUT OVERLAPS)
```

then no two suppliers could deliver the same product at the same time.

Inserting data into an application-period temporal table

Inserting data into an application-period temporal table is similar to inserting data into a regular table.

About this task

When inserting data into an application-period temporal table, the only special consideration is the need to include the row-begin and row-end columns that capture when the row is valid from the perspective of the associated business applications. This valid period is called the BUSINESS_TIME period. The database manager automatically generates an implicit check constraint that ensures that the begin column of the BUSINESS_TIME period is less than its end column. If a unique constraint or index with BUSINESS_TIME WITHOUT OVERLAPS was created for the table, you must ensure that no BUSINESS_TIME periods overlap.

Procedure

To insert data into an application-period temporal table, use the INSERT statement to add data to the table. For example, the following data was inserted to the table created in the example in Creating an application-period temporal table topic.

```
INSERT INTO policy_info VALUES('A123',12000,'2008-01-01','2008-07-01');

INSERT INTO policy_info VALUES('A123',16000,'2008-07-01','2009-01-01');

INSERT INTO policy_info VALUES('A123',16000,'2008-06-01','2008-08-01');

INSERT INTO policy_info VALUES('B345',18000,'2008-01-01','2009-01-01');

INSERT INTO policy_info VALUES('C567',20000,'2008-01-01','2009-01-01');
```

Results

There were five INSERT statements issued, but only four rows were added to the table. The second and third INSERT statements are attempting to add rows for policy A123, but their BUSINESS_TIME periods overlap which results in the following:

- The second insert adds a row for `policy_id` A123 with a `bus_start` value of 2008-07-01 and a `bus_end` value of 2009-01-01.
- The third insert attempts to add a row for `policy_id` A123, but it fails because its BUSINESS_TIME period overlaps that of the previous insert. The `policy_info` table was created with a BUSINESS_TIME WITHOUT OVERLAPS index and the third insert has a `bus_end` value of 2008-08-01, which is within the time period of the earlier insert.

The begin column of a period is inclusive, while the end column is exclusive, meaning that the row with a `bus_end` value of 2008-07-01 does not have a

BUSINESS_TIME period overlap with the row that contains a `bus_start` value of 2008-07-01. As a result, the `policy_info` table now contains the following insurance coverage data:

Table 44. Data added to an application-period temporal table (policy_info)

policy_id	coverage	bus_start	bus_end
A123	12000	2008-01-01	2008-07-01
A123	16000	2008-07-01	2009-01-01
B345	18000	2008-01-01	2009-01-01
C567	20000	2008-01-01	2009-01-01

Updating data in an application-period temporal table

Updating data in an application-period temporal table can be similar to updating data in a regular table, but data can also be updated for specified points of time in the past, present, or future. Point in time updates can result in rows being split and new rows being inserted automatically into the table.

About this task

In addition to the regular UPDATE statement, application-period temporal tables also support time range updates where the UPDATE statement includes the FOR PORTION OF BUSINESS_TIME clause. A row is a candidate for updating if its period-begin column, period-end column, or both fall within the range specified in the FOR PORTION OF BUSINESS_TIME clause.

Procedure

To update data in an application-period temporal table, use the UPDATE statement. For example, you discovered some errors in the insurance coverage information for some customers and the following updates are performed on the sample table that was introduced in the “ Inserting data into an application-period temporal table” topic.
The following table contains the original `policy_info` table data.

Table 45. Original data in the application-period temporal table (policy_info)

policy_id	coverage	bus_start	bus_end
A123	12000	2008-01-01	2008-07-01
A123	16000	2008-07-01	2009-01-01
B345	18000	2008-01-01	2009-01-01
C567	20000	2008-01-01	2009-01-01

The `policy_info` table was created with a BUSINESS_TIME WITHOUT OVERLAPS index. When using the regular UPDATE statement, you must ensure that no BUSINESS_TIME periods overlap. Updating an application-period temporal table by using the FOR PORTION OF BUSINESS_TIME clause avoids period overlap problems. This clause causes rows to be changed and can result in rows that are inserted when the existing time period for a row that is being updated is not fully contained within the range specified in the UPDATE statement.

- The coverage for policy B345 actually started on March 1, 2008 (2008-03-01) and the coverage should be 18500:

```
UPDATE policy_info
  SET coverage = 18500, bus_start = '2008-03-01'
  WHERE policy_id = 'B345'
  AND coverage=18000
```

The update to policy B345 uses a regular UPDATE statement. There is only one row in the `policy_info` table for `policy_id` B345, so there are no potential BUSINESS_TIME periods overlaps. As a result, the `bus_start` column value is updated to 2008-03-01 and the `coverage` value is updated to 18500. Note that updates to a BUSINESS_TIME period column cannot include the FOR PORTION OF BUSINESS_TIME clause.

Table 46. Policy B345 updated

policy_id	coverage	bus_start	bus_end
A123	12000	2008-01-01	2008-07-01
A123	16000	2008-07-01	2009-01-01
B345	18500	2008-03-01	2009-01-01
C567	20000	2008-01-01	2009-01-01

- The coverage for policy C567 should be 25000 for the year 2008:

```
UPDATE policy_info
   FOR PORTION OF BUSINESS_TIME FROM '2008-01-01' TO '2009-01-01'
   SET coverage = 25000
   WHERE policy_id = 'C567';
```

The update to policy C567 applies to the BUSINESS_TIME period from 2008-01-01 to 2009-01-01. There is only one row in the `policy_info` table for `policy_id` C567 that includes this time period. The BUSINESS_TIME period is fully contained within the `bus_start` and `bus_end` column values for that row. As a result, the `coverage` value is updated to 25000. The `bus_start` and `bus_end` column values are unchanged.

Table 47. Policy C567 updated

policy_id	coverage	bus_start	bus_end
A123	12000	2008-01-01	2008-07-01
A123	16000	2008-07-01	2009-01-01
B345	18500	2008-03-01	2009-01-01
C567	25000	2008-01-01	2009-01-01

- The coverage for policy A123 shows an increase from 12000 to 16000 on July 1 (2008-07-01), but an earlier increase to 14000 is missing:

```
UPDATE policy_info
   FOR PORTION OF BUSINESS_TIME FROM '2008-06-01' TO '2008-08-01'
   SET coverage = 14000
   WHERE policy_id = 'A123';
```

The update to policy A123 applies to the BUSINESS_TIME period from 2008-06-01 to 2008-08-01. There are two rows in the `policy_info` table for `policy_id` A123 that include part of this time period.

1. The BUSINESS_TIME period is partially contained in the row that has a `bus_start` value of 2008-01-01 and a `bus_end` value of 2008-07-01. This row overlaps the beginning of the specified period because the earliest time value in the BUSINESS_TIME period is greater than the rows `bus_start` value, but less than its `bus_end` value.

2. The BUSINESS_TIME period is partially contained in the row that has a `bus_start` value of 2008-07-01 and a `bus_end` value of 2009-01-01. This row overlaps the end of the specified period because the latest time value in the BUSINESS_TIME period is greater than the rows `bus_start` value, but less than its `bus_end` value.

As a result, the update causes the following things to occur:

1. When the `bus_end` value overlaps the beginning of the specified period, the row is updated to the new coverage value of 14000. In this updated row, the `bus_start` value is set to 2008-06-01 which is the begin value of the UPDATE specified period, and the `bus_end` value is unchanged. An additional row is inserted with the original values from the row, except that the `bus_end` value is set to 2008-06-01. This new row reflects the BUSINESS_TIME period when coverage was 12000.
2. When the `bus_start` value overlaps the end of the specified period, the row is updated to the new coverage value of 14000. In this updated row, the `bus_start` value is unchanged and the `bus_end` value is set to 2008-08-01 which is the end value of the UPDATE specified period. An additional row is inserted with the original values from the row, except that the `bus_start` value is set to 2008-08-01. This new row reflects the BUSINESS_TIME period when coverage was 16000.

Table 48. Policy A123 updated

policy_id	coverage	bus_start	bus_end
A123	12000	2008-01-01	2008-06-01
A123	14000	2008-06-01	2008-07-01
A123	14000	2008-07-01	2008-08-01
A123	16000	2008-08-01	2009-01-01
B345	18500	2008-03-01	2009-01-01
C567	25000	2008-01-01	2009-01-01

More examples

This section contains more updating application-period temporal table examples.

Merging content

In the following example, a MERGE statement uses the FOR PORTION OF clause to update the `policy_info` table with the contents of another table (`merge_policy`).

Table 49. Content of the merge_policy table

policy_id	coverage	bus_start	bus_end
C567	30000	2008-10-01	2010-05-01
H789	16000	2008-10-01	2010-05-01

1. Create global variables to hold the FROM and TO dates for the FOR PORTION OF clause.

```
CREATE VARIABLE sdate DATE default '2008-10-01';
CREATE VARIABLE edate DATE default '2010-05-01';
```

2. Issue a MERGE statement that merges the content of `merge_policy` into the `policy_info` table that resulted from the updates in the preceding “Procedure” section.

```
MERGE INTO policy_info pi1
   USING (SELECT policy_id, coverage, bus_start, bus_end
            FROM merge_policy) mp2
   ON (pi1.policy_id = mp2.policy_id)
WHEN MATCHED THEN
     UPDATE FOR PORTION OF BUSINESS_TIME FROM sdate TO edate
        SET pi1_coverage = mp2.coverage
WHEN NOT MATCHED THEN
      INSERT (policy_id, coverage, bus_start, bus_end)
       VALUES (mp2.policy_id, mp2.coverage, mp2.bus_start, mp2.bus_end)
```

The policy_id C567 is common to both tables. The C567 bus_start value in merge_policy overlaps the C567 bus_end value in policy_info. This statement results in the following items:

- The bus_end value for coverage of 25000 is set to 2008-10-01.
- A new row is inserted for coverage of 30000 with the bus_start and bus_end values from merge_policy.

The policy_id H789 exists only in merge_policy and so a new row is added to policy_info.

Table 50. Merged updated data in an application-period temporal table (policy_info)

policy_id	coverage	bus_start	bus_end
A123	12000	2008-01-01	2008-06-01
A123	14000	2008-06-01	2008-07-01
A123	14000	2008-07-01	2008-08-01
A123	16000	2008-08-01	2009-01-01
B345	18000	2008-03-01	2009-01-01
C567	25000	2008-01-01	2008-10-01
C567	30000	2008-10-01	2010-05-01
H789	16000	2008-10-01	2010-05-01

Update targets

The FOR PORTION OF BUSINESS_TIME clause can be used only when the target of the update statement is a table or a view. The following updates return errors.

```
UPDATE (SELECT * FROM policy_info) FOR PORTION OF BUSINESS_TIME
   FROM '2008-01-01' TO '06-15-2008' SET policy_id = policy_id + 1;

UPDATE (SELECT * FROM policy_info FOR BUSINESS_TIME AS OF '2008-01-01')
   FOR PORTION OF BUSINESS_TIME FROM '2008-01-01' TO '06-15-2008'
   SET policy_id = policy_id + 1;
```

Updating a view

A view with references to an application-period temporal table is updatable. The following UPDATE would update the policy_info table.

```
CREATE VIEW viewC AS SELECT * FROM policy_info;
UPDATE viewC SET coverage = coverage + 5000;
```

A view with an application-period temporal table in its FROM clause that contains a period specification is also updatable. This condition differs from views on system-period temporal tables and bitemporal tables.

```
CREATE VIEW viewD AS SELECT * FROM policy_info
   FOR BUSINESS_TIME AS OF CURRENT DATE;
UPDATE viewD SET coverage = coverage - 1000;
```

A FOR PORTION OF update clause can be included against views with references to application-period temporal tables or bitemporal tables. Such updates are propagated to the temporal tables referenced in the FROM clause of the view definition.

```
CREATE VIEW viewE AS SELECT * FROM policy_info;
UPDATE viewE FOR PORTION OF BUSINESS_TIME
   FROM '2009-01-01' TO '2009-06-01' SET coverage = coverage + 500;
```

Deleting data from an application-period temporal table

Deleting data from an application-period temporal table removes rows from the table and can potentially result in new rows that are inserted into the application-period temporal table itself.

About this task

In addition to the regular DELETE statement, application-period temporal tables also support time range deletes where the DELETE statement includes the FOR PORTION OF BUSINESS_TIME clause. A row is a candidate for deletion if its period-begin column, period-end column, or both fall within the range specified in the FOR PORTION OF BUSINESS_TIME clause.

Procedure

To delete data from an application-period temporal table, use the DELETE FROM statement to delete data. For example, it was discovered that policy A123 should not provide coverage from June 15, 2008 to August 15, 2008 and therefore that data should be deleted from the table that was updated in the Updating data in an application-period temporal table topic.

```
DELETE FROM policy_info
   FOR PORTION OF BUSINESS_TIME FROM '2008-06-15' TO '2008-08-15'
   WHERE policy_id = 'A123';
```

Results

The original `policy_info` table data is as follows:

Table 51. Data in the application-period temporal table (policy_info) before the DELETE statement

policy_id	coverage	bus_start	bus_end
A123	12000	2008-01-01	2008-06-01
A123	14000	2008-06-01	2008-07-01
A123	14000	2008-07-01	2008-08-01
A123	16000	2008-08-01	2009-01-01
B345	18000	2008-03-01	2009-01-01
C567	25000	2008-01-01	2009-01-01

Deleting data from an application-period temporal table by using the FOR PORTION OF BUSINESS_TIME clause causes rows to be deleted and can result in rows that are inserted when the time period for a row covers a portion of the range specified in the DELETE FROM statement. Deleting data related to policy A123 applies to the BUSINESS_TIME period from 2008-06-15 to 2008-08-15. There are three rows in the `policy_info` table for `policy_id` A123 that include all or part of that time period.

The update to policy A123 affects the system-period temporal table and its history table, causing the following the things to occur:

- There is one row where the BUSINESS_TIME period in the DELETE FROM statement covers the entire time period for a row. The row with a `bus_start` value of 2008-07-01 and a `bus_end` value of 2008-08-01 is deleted.
- When only the `bus_end` value falls into the specified period, the row is deleted. A new row is inserted with the original values from the deleted row, except that the `bus_end` value is set to 2008-06-15.
- When only the `bus_start` value falls into the specified period, the row is deleted. A new row is inserted with the original values from the deleted row, except that the `bus_start` value is set to 2008-08-15.

Table 52. Data in the application-period temporal table (policy_info) after the DELETE statement

policy_id	coverage	bus_start	bus_end
A123	12000	2008-01-01	2008-06-01
A123	14000	2008-06-01	2008-06-15
A123	16000	2008-08-15	2009-01-01
B345	18000	2008-03-01	2009-01-01
C567	25000	2008-01-01	2009-01-01

Example

This section contains more deleting application-period temporal table examples.

Delete targets

The FOR PORTION OF BUSINESS_TIME clause can be used only when the target of the delete statement is a table or a view. The following DELETE statement returns an error:

```
DELETE FROM (SELECT * FROM policy_info) FOR PORTION OF BUSINESS_TIME
   FROM '2008-01-01' TO '2008-06-15';
```

Querying application-period temporal data

Querying an application-period temporal table can return results for a specified time period.

About this task

When querying an application-period temporal table, you can include FOR BUSINESS_TIME in the FROM clause. Using FOR BUSINESS_TIME specifications, you can query the current, past, and future state of your data. Time periods are specified as follows:

AS OF *value1*

Includes all the rows where the begin value for the period is less than or equal to *value1* and the end value for the period is greater than *value1*.

FROM *value1* **TO** *value2*

Includes all the rows where the begin value for the period is greater than or equal to *value1* and the end value for the period is less than *value2*. This means that the begin time is included in the period, but the end time is not.

BETWEEN *value1* **AND** *value2*

Includes all the rows where any time period overlaps any point in time between *value1* and *value2*. A row is returned if the begin value for the period is less than or equal to *value2* and the end value for the period is greater than *value1*.

See the following section for some sample queries.

Procedure

To query an application-period temporal table, use the SELECT statement. For example, each of the following queries requests policy information for `policy_id` A123 from the result table in the " Updating data in an application-period temporal table" topic. Each query uses a variation of the time period specification. The `policy_info` table is as follows:

Table 53. Application-period temporal table: policy_info

policy_id	coverage	bus_start	bus_end
A123	12000	2008-01-01	2008-06-01
A123	14000	2008-06-01	2008-06-15
A123	16000	2008-08-15	2009-01-01
B345	18000	2008-03-01	2009-01-01
C567	25000	2008-01-01	2009-01-01

- Query with no time period specification. For example:

  ```
  SELECT policy_id, coverage, bus_start, bus_end
     FROM policy_info
     where policy_id = 'A123'
  ```

  This query returns all three rows for policy A123.

  ```
  A123, 12000, 2008-01-01, 2008-06-01
  A123, 14000, 2008-06-01, 2008-06-15
  A123, 16000, 2008-08-15, 2009-01-01
  ```

- Query with FOR BUSINESS_TIME AS OF specified. For example:

  ```
  SELECT policy_id, coverage, bus_start, bus_end
     FROM policy_info
     FOR BUSINESS_TIME AS OF '2008-07-15'
     where policy_id = 'A123'
  ```

  This query does not return any rows. There are no rows for A123 where the begin value for the period is less than or equal to *2008-07-15* and the end value for the period is greater than *2008-07-15*. Policy A123 had no coverage on *2008-07-15*.

- Query with FOR BUSINESS_TIME FROM...TO specified. For example:

  ```
  SELECT policy_id, coverage, bus_start, bus_end
     FROM policy_info
     FOR BUSINESS_TIME FROM
        '2008-01-01' TO '2008-06-15'
     where policy_id = 'A123'
  ```

  This query returns two rows. The begin-column of a period is inclusive, while the end-column is exclusive. The row with a bus_end value of 2008-06-15 is valid until 06-14-2008 at midnight and so is less than *value2*.

  ```
  A123, 12000, 2008-01-01, 2008-06-01
  A123, 14000, 2008-06-01, 2008-06-15
  ```

- Query with FOR BUSINESS_TIME BETWEEN...AND specified. For example:

  ```
  SELECT policy_id, coverage, bus_start, bus_end
     FROM policy_info
     FOR BUSINESS_TIME BETWEEN
        '0001-01-01' AND '2008-01-01'
  ```

  This query returns two rows. The rows with a bus_start value of 2008-01-01 are equal to *value1* and are returned because the begin time of a period is included. Note that if a row had a bus_end column value of 2008-01-01, that row would be returned because its end time is equal to *value1* and the end time of a period is included.

  ```
  A123, 12000, 2008-01-01, 2008-06-01
  C567, 25000, 2008-01-01, 2009-01-01
  ```

More examples

This section contains more querying application-period temporal table examples.

Querying a view

A view can be queried as if it were an application-period temporal table. Time period specifications (FOR BUSINESS_TIME) can be specified after the view reference.

```
CREATE VIEW policy_year_end(policy, amount, start_date, end_date)
   AS SELECT * FROM policy_info;

SELECT * FROM policy_year_end FOR BUSINESS_TIME AS OF '2008-12-31';
```

The SELECT on the view `policy_year_end` queries the `policy_info` table and returns all policies that were in effect at the end of 2008.

```
A123, 16000, 2008-08-15, 2009-01-01
B345, 18000, 2008-03-01, 2009-01-01
C567, 25000, 2008-01-01, 2009-01-01
```

If a view definition contains a period specification, then queries against the view cannot contain period specifications. The following statements return an error due to multiple period specifications:

```
CREATE VIEW all_policies AS SELECT * FROM policy_info;
   FOR BUSINESS_TIME AS OF '2008-02-28';

SELECT * FROM all_policies FOR BUSINESS_TIME BETWEEN
   FOR BUSINESS_TIME AS OF '2008-10-01';
```

Setting the application time for a session

Setting the application time in the CURRENT TEMPORAL BUSINESS_TIME special register can reduce or eliminate the changes required when running an application against different points in time.

About this task

When you have an application that you want to run against an application-period temporal table to query the state of your business for a number of different dates, you can set the date in a special register. If you need to query your data AS OF today, AS OF the end of the last quarter, or if you are simulating future events, AS OF some future date, it might not be possible to change the application and add AS OF specifications to each SQL statement. This restriction is likely the case when you are using packaged applications. To address such scenarios, you can use the CURRENT TEMPORAL BUSINESS_TIME special register to set the date at the session level.

Setting the CURRENT TEMPORAL BUSINESS_TIME special register does not affect regular tables. Only queries on temporal tables with a BUSINESS_TIME period enabled (application-period temporal tables and bitemporal tables) use the time set in the special register. There is also no affect on DDL statements.

Note: When the CURRENT TEMPORAL BUSINESS_TIME special register is set to a non-null value, data modification statements like INSERT, UPDATE, DELETE, and MERGE against application-period temporal tables are supported. This behavior differs from the CURRENT TEMPORAL SYSTEM_TIME special register which blocks data modification statements against system-period temporal table and bitemporal tables.

The setting for the BUSTIMESENSITIVE bind option determines whether references to application-period temporal tables and bitemporal tables in both static SQL statements and dynamic SQL statements in a package are affected by the value of the CURRENT TEMPORAL BUSINESS_TIME special register. The bind option can be set to YES or NO. For SQL procedures, use the SET_ROUTINE_OPTS procedure to set the bind-like options, called query compiler variables.

Procedure

When this special register is set to a non-null value, applications that issue a query returns data as of that date. The following examples request information from the result tables in the “ Deleting data from an application-period temporal table ”topic.

- Set the special register to a non-null value and query data as of that date. For example:

```
SET CURRENT TEMPORAL BUSINESS_TIME = '2008-01-01';
SELECT * FROM policy_info;
```

- Set the special register to a time and reference an application-period temporal table in view definitions.

```
CREATE VIEW view1 AS SELECT policy_id, coverage FROM policy_info;
CREATE VIEW view2 AS SELECT * FROM regular_table
   WHERE col1 IN (SELECT coverage FROM policy_info);
SET CURRENT TEMPORAL BUSINESS_TIME = '2008-01-01';
SELECT * FROM view1;
SELECT * FROM view2;
```

- Set the special register to a past date and issue a query that contains a time period specification. For example:

```
SET CURRENT TEMPORAL BUSINESS_TIME = CURRENT DATE - 1 YEAR;
SELECT * FROM policy_info FOR BUSINESS_TIME AS OF '2008-01-01';
```

Results

The `policy_info` table is as follows:

Table 54. Data in the application-period temporal table (policy_info) after the DELETE statement

policy_id	coverage	bus_start	bus_end
A123	12000	2008-01-01	2008-06-01

Table 54. Data in the application-period temporal table (policy_info) after the DELETE statement (continued)

policy_id	coverage	bus_start	bus_end
A123	14000	2008-06-01	2008-06-15
A123	16000	2008-08-15	2009-01-01
B345	18000	2008-03-01	2009-01-01
C567	25000	2008-01-01	2009-01-01

- The request for data as of 2008-01-01 queries the `policy_info` table. The query is implicitly rewritten to:

  ```
  SELECT * FROM policy_info FOR BUSINESS_TIME AS OF '2008-01-01';
  ```

  The query returns:

  ```
  A123, 12000, 2008-01-01, 2008-06-01
  C567, 25000, 2008-01-01, 2009-01-01
  ```

- The query on `view1` is implicitly rewritten to:

  ```
  SELECT * FROM view1 FOR BUSINESS_TIME AS OF CURRENT TEMPORAL BUSINESS_TIME;
  ```

  and then to:

  ```
  SELECT policy_id, coverage FROM policy_info
     FOR BUSINESS_TIME AS OF '2008-01-01';
  ```

  The query returns:

  ```
  A123, 12000
  C567, 25000
  ```

  The query on `view2` involves a view on a regular table that references an application-period temporal table, causing an implicit relationship between a regular table and the special register. The query is implicitly rewritten to:

  ```
  SELECT * FROM view2 FOR BUSINESS_TIME AS OF CURRENT TEMPORAL BUSINESS_TIME;
  ```

  and then to:

  ```
  SELECT * FROM regular_table WHERE col1 in (SELECT coverage FROM policy_info
     FOR BUSINESS_TIME AS OF '2008-01-01');
  ```

  The SELECT returns rows where `col1` values match values in `coverage`.

- An error is returned because there are multiple time period specifications. The special register was set to a non-null value and the query also specified a time.

Bitemporal tables

A bitemporal table is a table that combines the historical tracking of a system-period temporal table with the time-specific data storage capabilities of an application-period temporal table. Use bitemporal tables to keep user-based period information as well as system-based historical information.

Bitemporal tables behave as a combination of system-period temporal tables and application-period temporal tables. All the restrictions that apply to system-period temporal tables and application temporal tables also apply to bitemporal tables.

Creating a bitemporal table

Creating a bitemporal table results in a table that combines the historical tracking of a system-period temporal table with the time-specific data storage capabilities of an application-period temporal table.

About this task

When creating a bitemporal table, you combine the steps used to create a system-period temporal table with the steps used to create an application-period temporal table.

- Include both a SYSTEM_TIME period and a BUSINESS_TIME period in the CREATE TABLE statement.
- Create a history table to receive old rows from the bitemporal table.
- Add versioning to establish the link between the bitemporal table and the history table.
- Optionally, define that overlapping periods of BUSINESS_TIME are not allowed and that values are unique with respect to any period.

The examples in the following section show the creation of a table that stores policy information for the customers of an insurance company.

Procedure

To create a bitemporal table:

1. Create a table with both a SYSTEM_TIME attribute and a BUSINESS_TIME attribute. For example:

```
CREATE TABLE policy_info
(
 policy_id    CHAR(4) NOT NULL,
 coverage     INT NOT NULL,
 bus_start    DATE NOT NULL,
 bus_end      DATE NOT NULL,
 sys_start    TIMESTAMP(12) NOT NULL
              GENERATED ALWAYS AS ROW BEGIN,
 sys_end      TIMESTAMP(12) NOT NULL
              GENERATED ALWAYS AS ROW END,
 ts_id        TIMESTAMP(12) NOT NULL
              GENERATED ALWAYS AS TRANSACTION START ID,
 PERIOD BUSINESS_TIME (bus_start, bus_end),
 PERIOD SYSTEM_TIME (sys_start, sys_end)
) in policy_space;
```

2. Create a history table. For example:

```
CREATE TABLE hist_policy_info
(
 policy_id    CHAR(4) NOT NULL,
 coverage     INT NOT NULL,
 bus_start    DATE NOT NULL,
 bus_end      DATE NOT NULL,
 sys_start    TIMESTAMP(12) NOT NULL,
 sys_end      TIMESTAMP(12) NOT NULL,
 ts_id        TIMESTAMP(12)
) in hist_space;
```

You can also create a history table with the same names and descriptions as the columns of the system-period temporal table using the LIKE clause of the CREATE TABLE statement. For example:

```
CREATE TABLE hist_policy_info LIKE policy_info in hist_space;
```

3. Add versioning to the bitemporal table. For example:

```
ALTER TABLE policy_info ADD VERSIONING USE HISTORY TABLE hist_policy_info;
```

4. Optional: Create a unique index that includes the BUSINESS_TIME period. For example:

   ```
   CREATE UNIQUE INDEX ix_policy
      ON policy_info (policy_id, BUSINESS_TIME WITHOUT OVERLAPS);
   ```

Results

The policy_info table stores the insurance coverage level for a customer. The BUSINESS_TIME period-related columns (bus_start and bus_end) indicate when an insurance coverage level is valid. The SYSTEM_TIME period-related columns (sys_start and sys_end) show when a coverage level row is current. The ts_id column lists the time when execution started for a transaction that impacted the row.

Table 55. Created bitemporal table (policy_info)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id

The hist_policy_info history table receives the old rows from the policy_info table.

Table 56. Created history table (hist_policy_info)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id

The ix_policy index, with BUSINESS_TIME WITHOUT OVERLAPS as the final column in the index key column list, ensures that there are no overlapping time periods for customer insurance coverage levels.

Example

This section contains more creating bitemporal table examples.

Hiding columns

The following example creates the policy_info table with the TIMESTAMP(12) columns (sys_start, sys_end and ts_id) marked as implicitly hidden.

```
CREATE TABLE policy_info
(
 policy_id    CHAR(4) NOT NULL,
 coverage     INT NOT NULL,
 bus_start    DATE NOT NULL,
 bus_end      DATE NOT NULL,
 sys_start    TIMESTAMP(12) NOT NULL
              GENERATED ALWAYS AS ROW BEGIN IMPLICITLY HIDDEN,
 sys_end      TIMESTAMP(12) NOT NULL
              GENERATED ALWAYS AS ROW END IMPLICITLY HIDDEN,
 ts_id        TIMESTAMP(12)
              GENERATED ALWAYS AS TRANSACTION START ID IMPLICITLY HIDDEN,
 PERIOD BUSINESS_TIME (bus_start, bus_end),
 PERIOD SYSTEM_TIME   (sys_start, sys_end)
) in policy_space;
```

Creating the hist_policy_info history table using the LIKE clause of the CREATE TABLE statement results in the history table inheriting the implicitly hidden attribute from the policy_info table.

Inserting data into a bitemporal table

Inserting data into a bitemporal table is similar to inserting data into an application-period temporal table.

About this task

When inserting data into a bitemporal table, include begin and end columns that capture when the row is valid from the perspective of the associated business applications. This valid period is called the BUSINESS_TIME period. The database manager automatically generates an implicit check constraint that ensures that the begin column of the BUSINESS_TIME period is less than its end column. If a unique constraint or index with BUSINESS_TIME WITHOUT OVERLAPS was created for the table, this ensures that no BUSINESS_TIME periods overlap.

Procedure

To insert data into a bitemporal table, use the INSERT statement to add data to the table. For example, the following data was inserted on January 31, 2010 (2010-01-31) to the table created in the example in “ Creating a bitemporal table”.

```
INSERT INTO policy_info(policy_id, coverage, bus_start, bus_end)
   VALUES('A123',12000,'2008-01-01','2008-07-01');

INSERT INTO policy_info(policy_id, coverage, bus_start, bus_end)
   VALUES('A123',16000,'2008-07-01','2009-01-01');

INSERT INTO policy_info(policy_id, coverage, bus_start, bus_end)
   VALUES('B345',18000,'2008-01-01','2009-01-01');

INSERT INTO policy_info(policy_id, coverage, bus_start, bus_end)
   VALUES('C567',20000,'2008-01-01','2009-01-01');
```

Results

The policy_info table now contains the following insurance coverage data. The sys_start, sys_end, and ts_id column entries are generated by the database manager. The begin-column of a period is inclusive, while the end-column is exclusive, meaning that the row with a bus_end value of 2008-07-01 does not have a BUSINESS_TIME period overlap with the row that contains a bus_start value of 2008-07-01.

Table 57. Data added to a bitemporal table (policy_info)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-07-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
A123	16000	2008-07-01	2009-01-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
B345	18000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000

Table 57. Data added to a bitemporal table (policy_info) (continued)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
C567	20000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000

The hist_policy_info history table remains empty because no history rows are generated by an insert.

Table 58. History table (hist_policy_info) after insert

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id

Updating data in a bitemporal table

Updating data in a bitemporal table results in rows that are added to its associated history table and can potentially result in rows that are added to the bitemporal table itself.

About this task

In addition to the regular UPDATE statement, bitemporal tables also support time range updates where the UPDATE statement includes the FOR PORTION OF BUSINESS_TIME clause. A row is a candidate for updating if its period-begin column, period-end column, or both fall within the range specified in the FOR PORTION OF BUSINESS_TIME clause. Any existing impacted rows are copied to the history table before they are updated.

Procedure

To update data in a bitemporal table, use the UPDATE statement to change data rows. For example, it was discovered that there are some errors in the insurance coverage levels for two customers and the following data was updated on February 28, 2011 (2011-02-28) in the example table that had data added in the " Inserting data into a bitemporal table" topic.
The following table is the original policy_info table data.

Table 59. Original data in the bitemporal table (policy_info)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-07-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
A123	16000	2008-07-01	2009-01-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
B345	18000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
C567	20000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000

Updating a bitemporal table by using the FOR PORTION OF BUSINESS_TIME clause causes rows to be changed and can result in rows that are inserted when the existing time period for rows that are updated is not fully contained within the range specified in the UPDATE statement.

- The coverage for policy B345 actually started on March 1, 2008 (2008-03-01):

  ```
  UPDATE policy_info
    SET bus_start='2008-03-01'
    WHERE policy_id = 'B345'
    AND coverage = 18000;
  ```

  The update to policy B345 uses a regular UPDATE statement. There is only one row in the `policy_info` table for `policy_id` B345, so there are no potential BUSINESS_TIME periods overlaps. As a result the following things occur:

  1. The `bus_start` column value is updated to 2008-03-01. Note that updates to a BUSINESS_TIME period column cannot include the FOR PORTION OF BUSINESS_TIME clause.
  2. The database manager updates the `sys_start` and `ts_id` values to the date of the update.
  3. The original row is moved to the history table. The database manager updates the `sys_end` value to the date of the update.

  The following tables show the update for policy B345.

Table 60. Bitemporal table (policy_info) after policy B345 update

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-07-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
A123	16000	2008-07-01	2009-01-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
B345	18000	2008-03-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
C567	20000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000

Table 61. History table (hist_policy_info) after policy B345 update

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
B345	18000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000

- The coverage for policy C567 should be 25000 for the year 2008:

  ```
  UPDATE policy_info
     FOR PORTION OF BUSINESS_TIME FROM '2008-01-01' TO '2009-01-01'
     SET coverage = 25000
     WHERE policy_id = 'C567';
  ```

  The update to policy C567 applies to the BUSINESS_TIME period from 2008-01-01 to 2009-01-01. There is only one row in the `policy_info` table for `policy_id` C567 that includes that time period. The BUSINESS_TIME period is fully contained within the `bus_start` and `bus_end` column values for that row. As a result the following things occur:

  1. The coverage value for the row with `policy_id` C567 is updated to 25000.

2. The bus_start and bus_end column values are unchanged.
3. The database manager updates the sys_start and ts_id values to the date of the update.
4. The original row is moved to the history table. The database manager updates the sys_end value to the date of the update.

The following tables show the update for policy C567.

Table 62. Bitemporal table (policy_info) after policy C567 update

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-07-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
A123	16000	2008-07-01	2009-01-01	2010-01-31-22.31.33.495925000000	9999-12-30-00.00.00.000000000000	2010-01-31-22.31.33.495925000000
B345	18000	2008-03-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
C567	25000	2008-01-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

Table 63. History table (hist_policy_info) after policy C567 update

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
B345	18000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
C567	20000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000

- The coverage for policy A123 shows an increase from 12000 to 16000 on July 7 (2008-07-01), but an earlier increase to 14000 is missing:

```
UPDATE policy_info
   FOR PORTION OF BUSINESS_TIME FROM '2008-06-01' TO '2008-08-01'
   SET coverage = 14000
   WHERE policy_id = 'A123';
```

The update to policy A123 applies to the BUSINESS_TIME period from 2008-06-01 to 2008-08-01. There are two rows in the policy_info table for policy_id A123 that include part of the update time period.

1. The BUSINESS_TIME period is partially contained in the row that has a bus_start value of 2008-01-01 and a bus_end value of 2008-07-01. This row overlaps the beginning of the specified period because the earliest time value in the BUSINESS_TIME period is greater than the rows bus_start value, but less than its bus_end value.
2. The BUSINESS_TIME period is partially contained in the row that has a bus_start value of 2008-07-01 and a bus_end value of 2009-01-01. This row overlaps the end of the specified period because the latest time value in the BUSINESS_TIME period is greater than the rows bus_start value, but less than its bus_end value.

As a result the following things occur:

1. When the `bus_end` value overlaps the beginning of the specified period, the row is updated to the new coverage value of 14000. In this updated row, the `bus_start` value is set to 2008-06-01 which is the begin value of the UPDATE specified period, and the `bus_end` value is unchanged. An additional row is inserted with the original values from the row, except that the `bus_end` value is set to 2008-06-01. This new row reflects the BUSINESS_TIME period when coverage was 12000. The `sys_start`, `sys_end`, and `ts_id` column entries are generated by the database manager.
2. When the `bus_start` value overlaps the end of the specified period, the row is updated to the new coverage value of 14000. In this updated row, the `bus_start` value is unchanged and the `bus_end` value is set to 2008-08-01 which is the end value of the UPDATE specified period. An additional row is inserted with the original values from the row, except that the `bus_start` value is set to 2008-08-01. This new row reflects the BUSINESS_TIME period when coverage was 16000. The `sys_start`, `sys_end`, and `ts_id` column entries are generated by the database manager.
3. The original rows are moved to the history table. The database manager updates the `sys_end` value to the date of the update.

The following tables show the update for policy A123.

Table 64. Bitemporal table (policy_info) after policy A123 update

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-06-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
A123	14000	2008-06-01	2008-07-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
A123	14000	2008-07-01	2008-08-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
A123	16000	2008-08-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
B345	18000	2008-03-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
C567	25000	2008-01-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

Table 65. History table (hist_policy_info) after policy A123 update

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-07-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
A123	16000	2008-07-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
B345	18000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000

Table 65. History table (hist_policy_info) after policy A123 update (continued)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
C567	20000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000

Deleting data from a bitemporal table

Deleting data from a bitemporal table results in rows that are deleted from the table, rows that are added to its associated history table and can potentially result in new rows that are inserted into the bitemporal table itself.

About this task

In addition to the regular DELETE statement, bitemporal tables also support time range deletes where the DELETE statement includes the FOR PORTION OF BUSINESS_TIME clause. A row is a candidate for deletion if its period-begin column, period-end column, or both falls within the range specified in the FOR PORTION OF BUSINESS_TIME clause. Any existing impacted rows are copied to the history table before they are deleted.

Procedure

To delete data from a bitemporal table, use the DELETE FROM statement. For example, it was discovered that policy A123 did not have coverage from June 15, 2008 to August 15, 2008. The data was deleted on September 1, 2011 (2011-09-01) from the table that was updated in the “ Updating data in a bitemporal table” topic.

```
DELETE FROM policy_info
   FOR PORTION OF BUSINESS_TIME FROM '2008-06-15' TO '2008-08-15'
   WHERE policy_id = 'A123';
```

Results

The original `policy_info` table and `hist_policy_info` table data is as follows:

Table 66. Data in the bitemporal table (policy_info) before the DELETE statement

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-06-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
A123	14000	2008-06-01	2008-07-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
A123	14000	2008-07-01	2008-08-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
A123	16000	2008-08-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
B345	18000	2008-03-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

Table 66. Data in the bitemporal table (policy_info) before the DELETE statement (continued)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
C567	25000	2008-01-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

Table 67. Data in the history table (hist_policy_info) before the DELETE statement

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-07-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
A123	16000	2008-07-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
B345	18000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
C567	20000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000

Deleting data from a bitemporal table by using the FOR PORTION OF BUSINESS_TIME clause causes rows to be deleted and can result in rows that are inserted when the time period for a row covers a portion of the range specified in the DELETE FROM statement. Deleting data related to policy A123 applies to the BUSINESS_TIME period from 2008-06-15 to 2008-08-15. There are three rows in the `policy_info` table for `policy_id` A123 that include all or part of that time period.

As a result, the following things occur:

- There is one row where the BUSINESS_TIME period in the DELETE FROM statement covers the entire time period for a row. The row with a `bus_start` value of 2008-07-01 and a `bus_end` value of 2008-08-01 is deleted.
- When only the `bus_end` value falls into the specified period, the row is deleted. A new row is inserted with the original values from the deleted row, except that the `bus_end` value is set to 2008-06-15. The `sys_start`, `sys_end`, and `ts_id` column entries are generated by the database manager.
- When only the `bus_start` value falls into the specified period, the row is deleted. A new row is inserted with the original values from the deleted row, except that the `bus_start` value is set to 2008-08-15. The `sys_start`, `sys_end`, and `ts_id` column entries are generated by the database manager.
- The original rows are moved to the history table. The database manager updates the `sys_end` value to the date of the delete.

Table 68. Data in the bitemporal table (policy_info) after the DELETE statement

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-06-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

Table 68. Data in the bitemporal table (policy_info) after the DELETE statement (continued)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	14000	2008-06-01	2008-06-15	2011-09-01-12.18.22.959254000000	9999-12-30-00.00.00.000000000000	2011-09-01-12.18.22.959254000000
A123	16000	2008-08-15	2009-01-01	2011-09-01-12.18.22.959254000000	9999-12-30-00.00.00.000000000000	2011-09-01-12.18.22.959254000000
B345	18000	2008-03-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000
C567	25000	2008-01-01	2009-01-01	2011-02-28-09.10.12.649592000000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.649592000000

Table 69. History table (hist_policy_info) after DELETE statement

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-07-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
A123	16000	2008-07-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
B345	18000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
C567	20000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.649592000000	2010-01-31-22.31.33.495925000000
A123	14000	2008-06-01	2008-07-01	2011-02-28-09.10.12.649592000000	2011-09-01-12.18.22.959254000000	2011-09-01-12.18.22.959254000000
A123	14000	2008-07-01	2008-08-01	2011-02-28-09.10.12.649592000000	2011-09-01-12.18.22.959254000000	2011-09-01-12.18.22.959254000000
A123	16000	2008-08-01	2009-01-01	2011-02-28-09.10.12.649592000000	2011-09-01-12.18.22.959254000000	2011-09-01-12.18.22.959254000000

Querying bitemporal data

Querying a bitemporal table can return results for a specified time period. Those results can include current values, previous historic values, and future values.

About this task

When querying a bitemporal table, you can include FOR BUSINESS_TIME, FOR SYSTEM_TIME, or both in the FROM clause. Using these time period specifications, you can query the current, past, and future state of your data. Time periods are specified as follows:

AS OF *value1*

Includes all the rows where the begin value for the period is less than or equal to *value1* and the end value for the period is greater than *value1*. This enables you to query your data as of a certain point in time.

FROM *value1* **TO** *value2*
Includes all the rows where the begin value for the period is equal to or greater than *value1* and the end value for the period is less than *value2*. This means that the begin time is included in the period, but the end time is not.

BETWEEN *value1* **AND** *value2*
Includes all the rows where any time period overlaps any point in time between *value1* and *value2*. A row is returned if the begin value for the period is less than or equal to *value2* and the end value for the period is greater than *value1*.

See the following section for some sample queries.

Procedure

To query a bitemporal table, use the SELECT statement. For example, each of the following queries requests policy information for `policy_id` A123 from the result tables in the " Deleting data from a bitemporal table" topic. Each query uses a variation of the time period specification.
The `policy_info` table and its associated history table are as follows:

Table 70. Bitemporal table: policy_info

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-06-01	2011-02-28-09.10.12.64959200000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.64959200000
A123	14000	2008-06-01	2008-06-15	2011-09-01-12.18.22.959254000000	9999-12-30-00.00.00.000000000000	2011-09-01-12.18.22.959254000000
A123	16000	2008-08-15	2009-01-01	2011-09-01-12.18.22.959254000000	9999-12-30-00.00.00.000000000000	2011-09-01-12.18.22.959254000000
B345	18000	2008-03-01	2009-01-01	2011-02-28-09.10.12.64959200000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.64959200000
C567	25000	2008-01-01	2009-01-01	2011-02-28-09.10.12.64959200000	9999-12-30-00.00.00.000000000000	2011-02-28-09.10.12.64959200000

Table 71. History table: hist_policy_info

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
A123	12000	2008-01-01	2008-07-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.64959200000	2010-01-31-22.31.33.495925000000
A123	16000	2008-07-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.64959200000	2010-01-31-22.31.33.495925000000
B345	18000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.64959200000	2010-01-31-22.31.33.495925000000

Table 71. History table: hist_policy_info (continued)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
C567	20000	2008-01-01	2009-01-01	2010-01-31-22.31.33.495925000000	2011-02-28-09.10.12.64959200000	2010-01-31-22.31.33.495925000000
A123	14000	2008-06-01	2008-07-01	2011-02-28-09.10.12.64959200000	2011-09-01-12.18.22.959254000000	2011-09-01-12.18.22.959254000000
A123	14000	2008-07-01	2008-08-01	2011-02-28-09.10.12.64959200000	2011-09-01-12.18.22.959254000000	2011-09-01-12.18.22.959254000000
A123	16000	2008-08-01	2009-01-01	2011-02-28-09.10.12.64959200000	2011-09-01-12.18.22.959254000000	2011-09-01-12.18.22.959254000000

- Query with no time period specification. For example:

  ```
  SELECT policy_id, coverage, bus_start, bus_end
     FROM policy_info
     where policy_id = 'A123'
  ```

  This query returns three rows. The SELECT statement queries only the policy_info table. The history table is not queried because FOR SYSTEM_TIME was not specified.

  ```
  A123, 12000, 2008-01-01, 2008-06-01
  A123, 14000, 2008-06-01, 2008-06-15
  A123, 16000, 2008-08-15, 2009-01-01
  ```

- Query with FOR SYSTEM_TIME FROM...TO specified. For example:

  ```
  SELECT policy_id, coverage, bus_start, bus_end
     FROM policy_info
     FOR SYSTEM_TIME FROM
        '0001-01-01-00.00.00.000000' TO '9999-12-30-00.00.00.000000000000'
     where policy_id = 'A123'
  ```

  This query returns eight rows. The SELECT statement queries both the policy_info and the hist_policy_info tables.

  ```
  A123, 12000, 2008-01-01, 2008-06-01
  A123, 14000, 2008-06-01, 2008-06-15
  A123, 16000, 2008-08-15, 2009-01-01
  A123, 12000, 2008-01-01, 2008-07-01
  A123, 16000, 2008-07-01, 2009-01-01
  A123, 14000, 2008-06-01, 2008-07-01
  A123, 14000, 2008-07-01, 2008-08-01
  A123, 16000, 2008-08-01, 2009-01-01
  ```

- Query with FOR BUSINESS_TIME AS OF specified. For example:

  ```
  SELECT policy_id, coverage, bus_start, bus_end
     FROM policy_info
     FOR BUSINESS_TIME AS OF '2008-07-15'
     where policy_id = 'A123'
  ```

  This query does not return any rows. The SELECT statement queries only the policy_info table and there are no rows for A123 where the begin value for the period is less than or equal to *2008-07-15* and the end value for the period is greater than *2008-07-15*. Policy A123 had no coverage on 2008-07-15. The history table is not queried because FOR SYSTEM_TIME was not specified.

- Query with FOR BUSINESS_TIME AS OF and FOR SYSTEM_TIME FROM...TO specified. For example:

```
SELECT policy_id, coverage, bus_start, bus_end
   FROM policy_info
   FOR BUSINESS_TIME AS OF '2008-07-15'
   FOR SYSTEM_TIME FROM
      '0001-01-01-00.00.00.000000' TO '9999-12-30-00.00.00.000000000000'
   where policy_id = 'A123'
```

This query returns two rows. The SELECT queries both the policy_info and the hist_policy_info tables. The returned rows are found in the history table.

```
A123, 16000, 2008-07-01, 2009-01-01
A123, 14000, 2008-07-01, 2008-08-01
```

Chapter 21. Constraints

Within any business, data must often adhere to certain restrictions or rules. For example, an employee number must be unique. The database manager provides *constraints* as a way to enforce such rules.

The following types of constraints are available:

- NOT NULL constraints
- Unique (or unique key) constraints
- Primary key constraints
- Foreign key (or referential integrity) constraints
- (Table) Check constraints
- Informational constraints

Constraints are only associated with tables and are either defined as part of the table creation process (using the CREATE TABLE statement) or are added to a table's definition after the table has been created (using the ALTER TABLE statement). You can use the ALTER TABLE statement to modify constraints. In most cases, existing constraints can be dropped at any time; this action does not affect the table's structure or the data stored in it.

Note: Unique and primary constraints are only associated with table objects, they are often enforced through the use of one or more unique or primary key indexes.

Types of constraints

A *constraint* is a rule that is used for optimization purposes.

There are five types of constraints:

- A *NOT NULL constraint* is a rule that prevents null values from being entered into one or more columns within a table.
- A *unique constraint* (also referred to as a *unique key constraint*) is a rule that forbids duplicate values in one or more columns within a table. Unique and primary keys are the supported unique constraints. For example, a unique constraint can be defined on the supplier identifier in the supplier table to ensure that the same supplier identifier is not given to two suppliers.
- A *primary key constraint* is a column or combination of columns that has the same properties as a unique constraint. You can use a primary key and foreign key constraints to define relationships between tables.
- A *foreign key constraint* (also referred to as a *referential constraint* or a *referential integrity constraint*) is a logical rule about values in one or more columns in one or more tables. For example, a set of tables shares information about a corporation's suppliers. Occasionally, a supplier's name changes. You can define a referential constraint stating that the ID of the supplier in a table must match a supplier ID in the supplier information. This constraint prevents insert, update, or delete operations that would otherwise result in missing supplier information.
- A *(table) check constraint* (also called a *check constraint*) sets restrictions on data added to a specific table. For example, a table check constraint can ensure that the salary level for an employee is at least $20 000 whenever salary data is added or updated in a table containing personnel information.

An *informational constraint* is an attribute of a certain type of constraint, but one that is not enforced by the database manager.

NOT NULL constraints

NOT NULL constraints prevent null values from being entered into a column.

The null value is used in databases to represent an unknown state. By default, all of the built-in data types provided with the database manager support the presence of null values. However, some business rules might dictate that a value must always be provided (for example, every employee is required to provide emergency contact information). The NOT NULL constraint is used to ensure that a given column of a table is never assigned the null value. Once a NOT NULL constraint has been defined for a particular column, any insert or update operation that attempts to place a null value in that column will fail.

Because constraints only apply to a particular table, they are usually defined along with a table's attributes, during the table creation process. The following CREATE TABLE statement shows how the NOT NULL constraint would be defined for a particular column:

```
CREATE TABLE EMPLOYEES    (. . .
                           EMERGENCY_PHONE    CHAR(14)  NOT NULL,
                           . . .
                           );
```

Unique constraints

Unique constraints ensure that the values in a set of columns are unique and not null for all rows in the table. The columns specified in a unique constraint must be defined as NOT NULL. The database manager uses a unique index to enforce the uniqueness of the key during changes to the columns of the unique constraint.

Unique constraints can be defined in the CREATE TABLE or ALTER TABLE statement using the UNIQUE clause. For example, a typical unique constraint in a DEPARTMENT table might be that the department number is unique and not null.

Figure 22 shows that a duplicate record is prevented from being added to a table when a unique constraint exists for the table:

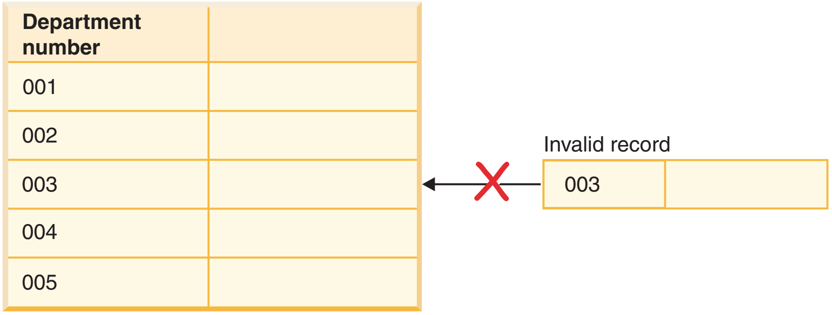


Figure 22. Unique constraints prevent duplicate data

The database manager enforces the constraint during insert and update operations, ensuring data integrity.

A table can have an arbitrary number of unique constraints, with at most one unique constraint defined as the primary key. A table cannot have more than one unique constraint on the same set of columns.

A unique constraint that is referenced by the foreign key of a referential constraint is called the *parent key*.

- When a unique constraint is defined in a CREATE TABLE statement, a unique index is automatically created by the database manager and designated as a primary or unique system-required index.
- When a unique constraint is defined in an ALTER TABLE statement and an index exists on the same columns, that index is designated as unique and system-required. If such an index does not exist, the unique index is automatically created by the database manager and designated as a primary or unique system-required index.

Note: There is a distinction between defining a unique constraint and creating a unique index. Although both enforce uniqueness, a unique index allows nullable columns and generally cannot be used as a parent key.

Primary key constraints

You can use primary key and foreign key constraints to define relationships between tables.

A primary key is a column or combination of columns that has the same properties as a unique constraint. Because the primary key is used to identify a row in a table, it must be unique, and must have the NOT NULL attribute. A table cannot have more than one primary key, but it can have multiple unique keys. Primary keys are optional, and can be defined when a table is created or altered. They are also beneficial, because they order the data when data is exported or reorganized.

(Table) Check constraints

A *check constraint* (also referred to as a *table check constraint*) is a database rule that specifies the values allowed in one or more columns of every row of a table. Specifying check constraints is done through a restricted form of a search condition.

Designing check constraints

When creating check constraints, one of two things can happen: (i) all the rows meet the check constraint, or (ii) some or all the rows do not meet the check constraint.

About this task

All the rows meet the check constraint

When all the rows meet the check constraint, the check constraint will be created successfully. Future attempts to insert or update data that does not meet the constraint business rule will be rejected.

Some or all the rows do not meet the check constraint

When there are some rows that do not meet the check constraint, the check constraint will not be created (that is, the ALTER TABLE statement will fail). The ALTER TABLE statement, which adds a new constraint to the EMPLOYEE table, is shown in the following example. The check constraint is named CHECK_JOB. The database manager will use this name to inform you about which constraint was violated if an INSERT or UPDATE statement fails. The CHECK clause is used to define a table-check constraint.

```
ALTER TABLE EMPLOYEE
  ADD CONSTRAINT check_job
  CHECK (JOB IN ('Engineer', 'Sales', 'Manager'));
```

An ALTER TABLE statement was used because the table had already been defined. If there are values in the EMPLOYEE table that conflict with the constraint being defined, the ALTER STATEMENT will not be completed successfully.

As check constraints and other types of constraints are used to implement business rules, you might need to change them from time to time. This could happen when the business rules change in your organization. Whenever a check constraint needs to be changed, you must drop it and re-create a new one. Check constraints can be dropped at any time, and this action will not affect your table or the data within it. When you drop a check constraint, you must be aware that data validation performed by the constraint will no longer be in effect.

Comparison of check constraints and BEFORE triggers

You must consider the difference between check constraints when considering whether to use triggers or check constraints to preserve the integrity of your data.

The integrity of the data in a relational database must be maintained as multiple users access and change the data. Whenever data is shared, there is a need to ensure the accuracy of the values within databases.

Check constraints

A (table) check constraint sets restrictions on data added to a specific table. You can use a table check constraint to define restrictions, beyond those of the data type, on the values that are allowed for a column in the table. Table check constraints take the form of range checks or checks against other values in the same row of the same table.

If the rule applies for all applications that use the data, use a table check constraint to enforce your restriction on the data allowed in the table. Table check constraints make the restriction generally applicable and easier to maintain.

The enforcement of check constraints is important for maintaining data integrity, but it also carries a certain amount of system activity that can impact performance whenever large volumes of data are modified.

BEFORE triggers

By using triggers that run before an update or insert, values that are being updated or inserted can be modified *before* the database is actually modified. These can be used to transform input from the application (user view of the data) to an internal database format where desired. BEFORE triggers can also be used to cause other non-database operations to be activated through user-defined functions.

In addition to modification, a common use of the BEFORE triggers is for data verification using the SIGNAL clause.

There are two differences between BEFORE triggers and check constraints when used for data verification:

1. BEFORE triggers, unlike check constraints, are not restricted to access other values in the same row of the same table.
2. During a SET INTEGRITY operation on a table after a LOAD operation, triggers (including BEFORE triggers) are not executed. Check constraints, however, are verified.

Foreign key (referential) constraints

Foreign key constraints (also known as *referential constraints* or *referential integrity constraints*) enable you to define required relationships between and within tables.

For example, a typical foreign key constraint might state that every employee in the EMPLOYEE table must be a member of an existing department, as defined in the DEPARTMENT table.

Referential integrity is the state of a database in which all values of all foreign keys are valid. A *foreign key* is a column or a set of columns in a table whose values are required to match at least one primary key or unique key value of a row in its parent table. A *referential constraint* is the rule that the values of the foreign key are valid only if one of the following conditions is true:
- They appear as values of a parent key.
- Some component of the foreign key is null.

To establish this relationship, you would define the department number in the EMPLOYEE table as the foreign key, and the department number in the DEPARTMENT table as the primary key.

Figure 23 on page 304 shows how a record with an invalid key is prevented from being added to a table when a foreign key constraint exists between two tables:

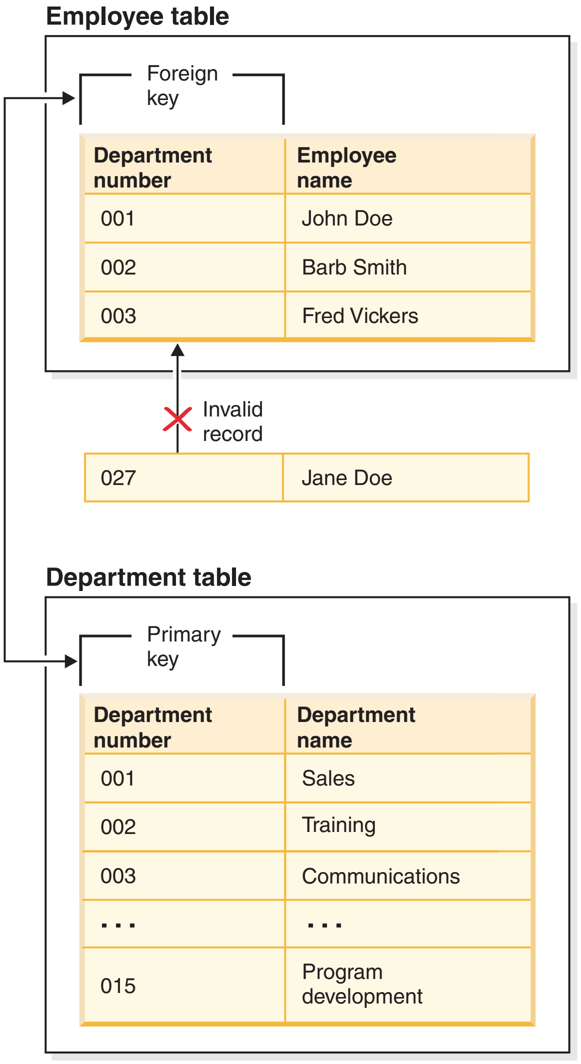


Figure 23. Foreign and primary key constraints

The table containing the parent key is called the *parent table* of the referential constraint, and the table containing the foreign key is said to be a *dependent* of that table.

Referential constraints can be defined in the CREATE TABLE statement or the ALTER TABLE statement. Referential constraints are enforced by the database manager during the execution of INSERT, UPDATE, DELETE, ALTER TABLE, MERGE, ADD CONSTRAINT, and SET INTEGRITY statements.

Referential integrity rules involve the following terms:

Table 72. Referential integrity terms

Concept	Terms
Parent key	A primary key or a unique key of a referential constraint.
Parent row	A row that has at least one dependent row.
Parent table	A table that contains the parent key of a referential constraint. A table can be a parent in an arbitrary number of referential constraints. A table that is the parent in a referential constraint can also be the dependent in a referential constraint.

Table 72. Referential integrity terms (continued)

Concept	Terms
Dependent table	A table that contains at least one referential constraint in its definition. A table can be a dependent in an arbitrary number of referential constraints. A table that is the dependent in a referential constraint can also be the parent in a referential constraint.
Descendent table	A table is a descendent of table T if it is a dependent of T or a descendent of a dependent of T.
Dependent row	A row that has at least one parent row.
Descendent row	A row is a descendent of row r if it is a dependent of r or a descendent of a dependent of r.
Referential cycle	A set of referential constraints such that each table in the set is a descendent of itself.
Self-referencing table	A table that is a parent and a dependent in the same referential constraint. The constraint is called a *self-referencing constraint*.
Self-referencing row	A row that is a parent of itself.

The purpose of a referential constraint is to guarantee that table relationships are maintained and that data entry rules are followed. This means that as long as a referential constraint is in effect, the database manager guarantees that for each row in a child table that has a non-null value in its foreign key columns, a row exists in a corresponding parent table that has a matching value in its parent key.

When an SQL operation attempts to change data in such a way that referential integrity will be compromised, a foreign key (or referential) constraint could be violated. The database manager handles these types of situations by enforcing a set of rules that are associated with each referential constraint. This set of rules consist of:

- An insert rule
- An update rule
- A delete rule

When an SQL operation attempts to change data in such a way that referential integrity will be compromised, a referential constraint could be violated. For example,

- An insert operation could attempt to add a row of data to a child table that has a value in its foreign key columns that does not match a value in the corresponding parent table's parent key.
- An update operation could attempt to change the value in a child table's foreign key columns to a value that has no matching value in the corresponding parent table's parent key.
- An update operation could attempt to change the value in a parent table's parent key to a value that does not have a matching value in a child table's foreign key columns.
- A delete operation could attempt to remove a record from a parent table that has a matching value in a child table's foreign key columns.

The database manager handles these types of situations by enforcing a set of rules that are associated with each referential constraint. This set of rules consists of:

- An insert rule

- An update rule
- A delete rule

Insert rule

The insert rule of a referential constraint is that a non-null insert value of the foreign key must match some value of the parent key of the parent table. The value of a composite foreign key is null if any component of the value is null. This rule is implicit when a foreign key is specified.

Update rule

The update rule of a referential constraint is specified when the referential constraint is defined. The choices are NO ACTION and RESTRICT. The update rule applies when a row of the parent or a row of the dependent table is updated.

In the case of a parent row, when a value in a column of the parent key is updated, the following rules apply:
- If any row in the dependent table matches the original value of the key, the update is rejected when the update rule is RESTRICT.
- If any row in the dependent table does not have a corresponding parent key when the update statement is completed (excluding AFTER triggers), the update is rejected when the update rule is NO ACTION.

The value of the parent unique keys cannot be changed if the update rule is RESTRICT and there are one or more dependent rows. However, if the update rule is NO ACTION, parent unique keys can be updated as long as every child has a parent key by the time the update statement completes. A non-null update value of a foreign key must be equal to a value of the primary key of the parent table of the relationship.

Also, the use of NO ACTION or RESTRICT as update rules for referential constraints determines when the constraint is enforced. An update rule of RESTRICT is enforced before all other constraints, including those referential constraints with modifying rules such as CASCADE or SET NULL. An update rule of NO ACTION is enforced after other referential constraints. Note that the SQLSTATE returned is different depending on whether the update rule is RESTRICT or NO ACTION.

In the case of a dependent row, the NO ACTION update rule is implicit when a foreign key is specified. NO ACTION means that a non-null update value of a foreign key must match some value of the parent key of the parent table when the update statement is completed.

The value of a composite foreign key is null if any component of the value is null.

Delete rule

The delete rule of a referential constraint is specified when the referential constraint is defined. The choices are NO ACTION, RESTRICT, CASCADE, or SET NULL. SET NULL can be specified only if some column of the foreign key allows null values.

If the identified table or the base table of the identified view is a parent, the rows selected for delete must not have any dependents in a relationship with a delete rule of RESTRICT, and the DELETE must not cascade to descendent rows that have dependents in a relationship with a delete rule of RESTRICT.

If the delete operation is not prevented by a RESTRICT delete rule, the selected rows are deleted. Any rows that are dependents of the selected rows are also affected:

- The nullable columns of the foreign keys of any rows that are their dependents in a relationship with a delete rule of SET NULL are set to the null value.
- Any rows that are their dependents in a relationship with a delete rule of CASCADE are also deleted, and the rules mentioned previously apply, in turn, to those rows.

The delete rule of NO ACTION is checked to enforce that any non-null foreign key refers to an existing parent row after the other referential constraints have been enforced.

The delete rule of a referential constraint applies only when *a row* of the parent table is deleted. More precisely, the rule applies only when *a row* of the parent table is the object of a delete or propagated delete operation (defined in the following section), and that row has dependents in the dependent table of the referential constraint. Consider an example where P is the parent table, D is the dependent table, and p is a parent row that is the object of a delete or propagated delete operation. The delete rule works as follows:

- With RESTRICT or NO ACTION, an error occurs and no rows are deleted.
- With CASCADE, the delete operation is propagated to the dependents of p in table D.
- With SET NULL, each nullable column of the foreign key of each dependent of p in table D is set to null.

Any table that can be involved in a delete operation on P is said to be *delete-connected* to P. Thus, a table is delete-connected to table P if it is a dependent of P, or a dependent of a table to which delete operations from P cascade.

The following restrictions apply to delete-connected relationships:

- When a table is delete-connected to itself in a referential cycle of more than one table, the cycle must not contain a delete rule of either RESTRICT or SET NULL.
- A table must not both be a dependent table in a CASCADE relationship (self-referencing or referencing another table) and have a self-referencing relationship with a delete rule of either RESTRICT or SET NULL.
- When a table is delete-connected to another table through multiple relationships where such relationships have overlapping foreign keys, these relationships must have the same delete rule and none of these can be SET NULL.
- When a table is delete-connected to another table through multiple relationships where one of the relationships is specified with delete rule SET NULL, the foreign key definition of this relationship must not contain any distribution key or MDC key column, a table-partitioning key column, or RCT key column.
- When two tables are delete-connected to the same table through CASCADE relationships, the two tables must not be delete-connected to each other where the delete connected paths end with delete rule RESTRICT or SET NULL.

Examples of interaction between triggers and referential constraints

Update operations can cause the interaction of triggers with referential constraints and check constraints.

Figure 24 and the associated description are representative of the processing that is performed for an statement that updates data in the database.

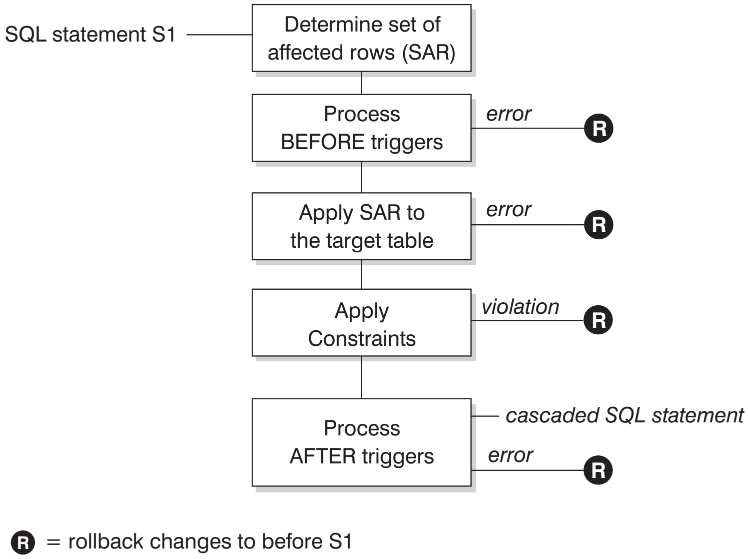


Figure 24. Processing an statement with associated triggers and constraints

Figure 24 shows the general order of processing for an statement that updates a table. It assumes a situation where the table includes BEFORE triggers, referential constraints, check constraints and AFTER triggers that cascade. The following is a description of the boxes and other items found in Figure 24.

- statement S_1

  This is the DELETE, INSERT, or UPDATE statement that begins the process. The statement S_1 identifies a table (or an updatable view over some table) referred to as the *subject table* throughout this description.

- Determine set of affected rows

  This step is the starting point for a process that repeats for referential constraint delete rules of CASCADE and SET NULL and for cascaded statements from AFTER triggers.

  The purpose of this step is to determine the *set of affected rows* for the statement. The set of rows included is based on the statement:

  – for DELETE, all rows that satisfy the search condition of the statement (or the current row for a positioned DELETE)

  – for INSERT, the rows identified by the VALUES clause or the fullselect

  – for UPDATE, all rows that satisfy the search condition (or the current row for a positioned UPDATE).

  If the set of affected rows is empty, there will be no BEFORE triggers, changes to apply to the subject table, or constraints to process for the statement.

- Process BEFORE triggers

All BEFORE triggers are processed in ascending order of creation. Each BEFORE trigger will process the triggered action once for each row in the set of affected rows.

An error can occur during the processing of a triggered action in which case all changes made as a result of the original statement S_1 (so far) are rolled back.

If there are no BEFORE triggers or the set of affected is empty, this step is skipped.

- Apply the set of affected rows to the subject table

  The actual delete, insert, or update is applied using the set of affected rows to the subject table in the database.

  An error can occur when applying the set of affected rows (such as attempting to insert a row with a duplicate key where a unique index exists) in which case all changes made as a result of the original statement S_1 (so far) are rolled back.

- Apply Constraints

  The constraints associated with the subject table are applied if set of affected rows is not empty. This includes unique constraints, unique indexes, referential constraints, check constraints and checks related to the WITH CHECK OPTION on views. Referential constraints with delete rules of cascade or set null might cause additional triggers to be activated.

  A violation of any constraint or WITH CHECK OPTION results in an error and all changes made as a result of S_1 (so far) are rolled back.

- Process AFTER triggers

  All AFTER triggers activated by S_1 are processed in ascending order of creation.

  FOR EACH STATEMENT triggers will process the triggered action exactly once, even if the set of affected rows is empty. FOR EACH ROW triggers will process the triggered action once for each row in the set of affected rows.

  An error can occur during the processing of a triggered action in which case all changes made as a result of the original S_1 (so far) are rolled back.

  The triggered action of a trigger can include triggered statements that are DELETE, INSERT or UPDATE statements. For the purposes of this description, each such statement is considered a *cascaded statement*.

  A cascaded statement is a DELETE, INSERT, or UPDATE statement that is processed as part of the triggered action of an AFTER trigger. This statement starts a cascaded level of trigger processing. This can be thought of as assigning the triggered statement as a new S_1 and performing all of the steps described here recursively.

  Once all triggered statements from all AFTER triggers activated by each S_1 have been processed to completion, the processing of the original S_1 is completed.

- R = roll back changes to before S_1

  Any error (including constraint violations) that occurs during processing results in a roll back of all the changes made directly or indirectly as a result of the original statement S_1. The database is therefore back in the same state as immediately before the execution of the original statement S_1

Informational constraints

An *informational constraint* is a constraint attribute that can be used by the SQL compiler to improve the access to data. Informational constraints are not enforced by the database manager, and are not used for additional verification of data; rather, they are used to improve query performance.

Informational constraints are defined using the CREATE TABLE or ALTER TABLE statements. You first add referential integrity or check constraints and then associate constraint attributes to them specifying whether the database manager is to enforce the constraint or not. For check constraints you can further specify that the constraint can be trusted. For referential integrity constraints, if the constraint is not enforced, you can further specify whether the constraint can be trusted or not. A not enforced and not trusted constraint is also known as a statistical referential integrity constraint. After you have specified the constraint you can then specify whether the constraint is to be used for query optimization or not.

Informational RI (referential integrity) constraints are used to optimize query performance, the incremental processing of REFRESH IMMEDIATE MQT, and staging tables. Query results, MQT data, and staging tables might be incorrect if informational constraints are violated.

For example, the order in which parent-child tables are maintained is important. When you want to add rows to a parent-child table, you must insert rows into the parent table first. To remove rows from a parent-child table, you must delete rows from the child table first. This ensures that there are no orphan rows in the child table at any time. Otherwise the informational constraint violation might affect the correctness of queries being executed during table maintenance, as well as the correctness of the incremental maintenance of dependent MQT data and staging tables.

Designing informational constraints

Constraints that are enforced by the database manager when records are inserted or updated can lead to high amounts of system activity, especially when loading large quantities of records that have referential integrity constraints. If an application has already verified information before inserting a record into the table, it might be more efficient to use informational constraints, rather than normal constraints.

Informational constraints tell the database manager what rules the data conforms to, but the rules are not enforced by the database manager. However, this information can be used by the DB2 optimizer and could result in better performance of SQL queries.

The following example illustrates the use of information constraints and how they work. This simple table contains information about applicants' age and gender:

```
CREATE TABLE APPLICANTS
  (
   AP_NO INT NOT NULL,
     GENDER  CHAR(1) NOT NULL,
CONSTRAINT GENDEROK
   CHECK (GENDER IN ('M', 'F'))
   NOT ENFORCED
   ENABLE QUERY OPTIMIZATION,
      AGE  INT NOT NULL,
CONSTRAINT AGEOK
   CHECK (AGE BETWEEN 1 AND 80)
   NOT ENFORCED
   ENABLE QUERY OPTIMIZATION,
 );
```

This example contains two options that change the behavior of the column constraints. The first option is NOT ENFORCED, which instructs the database manager not to enforce the checking of this column when data is inserted or updated. This option can be further specified to be either TRUSTED or NOT TRUSTED. If the informational constraint is specified to be TRUSTED then the database manager can trust that the data will conform to the constraint. This is the default option. If NOT TRUSTED is specified then the database manager knows that most of the data, but not all, will not conform to the constraint. In this example, the option is NOT ENFORCED TRUSTED by default since the option of trusted or not trusted was not specified.

The second option is ENABLE QUERY OPTIMIZATION which is used by the database manager when SELECT statements are run against this table. When this value is specified, the database manager will use the information in the constraint when optimizing the SQL.

If the table contains the NOT ENFORCED option, the behavior of insert statements might appear odd. The following SQL will not result in any errors when run against the APPLICANTS table:

```
INSERT INTO APPLICANTS VALUES
 (1, 'M', 54),
 (2, 'F', 38),
 (3, 'M', 21),
 (4, 'F', 89),
 (5, 'C', 10),
 (6, 'S',100),
```

Applicant number five has a gender (C), for child, and applicant number six has both an unusual gender and exceeds the age limits of the AGE column. In both cases the database manager will allow the insert to occur since the constraints are NOT ENFORCED and TRUSTED. The result of a select statement against the table is shown in the following example:

```
SELECT * FROM APPLICANTS
  WHERE GENDER = 'C';

APPLICANT  GENDER  AGE
---------  ------  ---

0 record(s) selected.
```

The database manager returned the incorrect answer to the query, even though the value 'C' is found within the table, but the constraint on this column tells the database manager that the only valid values are either 'M' or 'F'. The ENABLE QUERY OPTIMIZATION keyword also allowed the database manager to use this constraint information when optimizing the statement. If this is not the behavior that you want, then the constraint needs to be changed through the use of the ALTER TABLE statement, as shown in the following example:

```
ALTER TABLE APPLICANTS
  ALTER CHECK AGEOK DISABLE QUERY OPTIMIZATION
```

If the query is reissued, the database manager will return the following correct results:

```
SELECT * FROM APPLICANTS
  WHERE SEC = 'C';

APPLICANT GENDER  AGE
---------  ------  ---
        5  C       10

1 record(s) selected.
```

Note: If the constraint attributes NOT ENFORCED NOT TRUSTED and ENABLE QUERY OPTIMIZATION were specified from the beginning for the table APPLICANTS, then the correct results shown previously would have been returned after the first SELECT statement was issued.

The best scenario for using NOT ENFORCED TRUSTED informational constraints occurs when you can guarantee that the application program is the only application inserting and updating the data. If the application already checks all of the information beforehand (such as gender and age in the previous example) then using informational constraints can result in faster performance and no duplication of effort. Another possible use of informational constraints is in the design of data warehouses. Also, if you cannot guarantee that the data in the table will always conform to the constraint you can set the constraints to be NOT ENFORCED and NOT TRUSTED. This type of constraint can be used when strict matching between the values in the foreign keys and the primary keys are not needed. This constraint can also still be used as part of a statistical view enabling the optimization of certain SQL queries.

Creating and modifying constraints

Constraints can be added to existing tables with the ALTER TABLE statement.

About this task

The constraint name cannot be the same as any other constraint specified within an ALTER TABLE statement, and must be unique within the table (this includes the names of any referential integrity constraints that are defined). Existing data is checked against the new condition before the statement succeeds.

Creating and modifying unique constraints

Unique constraints can be added to an existing table. The constraint name cannot be the same as any other constraint specified within the ALTER TABLE statement, and must be unique within the table (this includes the names of any referential integrity constraints that are defined). Existing data is checked against the new condition before the statement succeeds.

To define unique constraints using the command line, use the `ADD CONSTRAINT` option of the ALTER TABLE statement. For example, the following statement adds a unique constraint to the EMPLOYEE table that represents a new way to uniquely identify employees in the table:

```
ALTER TABLE EMPLOYEE
   ADD CONSTRAINT NEWID UNIQUE(EMPNO,HIREDATE)
```

To modify this constraint, you would have to drop it, and then re-create it.

Creating and modifying primary key constraints

A primary key constraint can be added to an existing table. The constraint name must be unique within the table (this includes the names of any referential integrity constraints that are defined). Existing data is checked against the new condition before the statement succeeds.

To add primary keys using the command line, enter:

```
ALTER TABLE <name>
   ADD CONSTRAINT <column_name>
   PRIMARY KEY <column_name>
```

An existing constraint cannot be modified. To define another column, or set of columns, as the primary key, the existing primary key definition must first be dropped, and then re-created.

Creating and modifying check constraints

When a table check constraint is added, packages and cached dynamic SQL that insert or update the table might be marked as invalid.

To add a table check constraint using the command line, enter:

```
ALTER TABLE EMPLOYEE
   ADD CONSTRAINT REVENUE CHECK (SALARY + COMM > 25000)
```

To modify this constraint, you would have to drop it, and then re-create it.

Creating and modifying foreign key (referential) constraints

A foreign key is a reference to the data values in another table. There are different types of foreign key constraints.

When a foreign key is added to a table, packages and cached dynamic SQL containing the following statements might be marked as invalid:

- Statements that insert or update the table containing the foreign key
- Statements that update or delete the parent table.

To add foreign keys using the command line, enter:

```
ALTER TABLE <name>
   ADD CONSTRAINT <column_name>
    FOREIGN KEY <column_name>
    ON DELETE <action_type>
    ON UPDATE <action_type>
```

The following examples show the ALTER TABLE statement to add primary keys and foreign keys to a table:

```
ALTER TABLE PROJECT
  ADD CONSTRAINT PROJECT_KEY
      PRIMARY KEY (PROJNO)
ALTER TABLE EMP_ACT
  ADD CONSTRAINT ACTIVITY_KEY
      PRIMARY KEY (EMPNO, PROJNO, ACTNO)
  ADD CONSTRAINT ACT_EMP_REF
      FOREIGN KEY (EMPNO)
      REFERENCES EMPLOYEE
      ON DELETE  RESTRICT
  ADD CONSTRAINT ACT_PROJ_REF
      FOREIGN KEY (PROJNO)
      REFERENCES PROJECT
      ON DELETE CASCADE
```

To modify this constraint, you would have to drop it and then re-create it.

Creating and modifying informational constraints

To improve the performance of queries, you can add informational constraints to your tables. You add informational constraints using the CREATE TABLE or ALTER TABLE statement when you specify the `NOT ENFORCED` option on the DDL. Along with the NOT ENFORCED option you can further specify the constraint to be either TRUSTED or NOT TRUSTED.

Restriction: After you define informational constraints on a table, you can only alter the column names for that table after you remove the informational constraints.

To specify informational constraints on a table using the command line, enter one of the following commands for a new table:

```
ALTER TABLE <name> <constraint attributes> NOT ENFORCED
```

```
ALTER TABLE <name> <constraint attributes> NOT ENFORCED TRUSTED
```

```
ALTER TABLE <name> <constraint attributes> NOT ENFORCED NOT TRUSTED
```

ENFORCED or NOT ENFORCED: Specifies whether the constraint is enforced by the database manager during normal operations such as insert, update, or delete.

- ENFORCED cannot be specified for a functional dependency (SQLSTATE 42621).
- NOT ENFORCED should only be specified if the table data is independently known to conform to the constraint. Query results might be unpredictable if the data does not actually conform to the constraint. You can also specify if the NOT ENFORCED constraint is to be TRUSTED or NOT TRUSTED.
  - TRUSTED: Informs the database manager that the data can be trusted to conform to the constraint. This is the default option. This option must only be used if the data is independently known to conform to the constraint
  - NOT TRUSTED: Informs the database manager that the data cannot be trusted to conform to the constraint. This option is intended for cases where the data conforms to the constraint for most rows, but it is not independently known to conform to the constraint. NOT TRUSTED can be specified only for referential integrity constraints (SQLSTATE 42613).

To modify this constraint, you would have to drop it and then re-create it.

Table constraint implications for utility operations

If the table being loaded into has referential integrity constraints, the load utility places the table into the set integrity pending state to inform you that the SET INTEGRITY statement is required to be run on the table, in order to verify the referential integrity of the loaded rows. After the load utility has completed, you will need to issue the SET INTEGRITY statement to carry out the referential integrity checking on the loaded rows and to bring the table out of the set integrity pending state.

For example, if the DEPARTMENT and EMPLOYEE tables are the only tables that have been placed in set integrity pending state, you can execute the following statement:

```
SET INTEGRITY FOR DEPARTMENT ALLOW WRITE ACCESS,
   EMPLOYEE ALLOW WRITE ACCESS,
   IMMEDIATE CHECKED FOR EXCEPTION IN DEPARTMENT,
   USE DEPARTMENT_EX,
   IN EMPLOYEE USE EMPLOYEE_EX
```

The import utility is affected by referential constraints in the following ways:

- The REPLACE and REPLACE CREATE functions are not allowed if the object table has any dependents other than itself.

  To use these functions, first drop all foreign keys in which the table is a parent. When the import is complete, re-create the foreign keys with the ALTER TABLE statement.

- The success of importing into a table with self-referencing constraints depends on the order in which the rows are imported.

Statement dependencies when changing objects

Statement dependencies include package and cached dynamic SQL and XQuery statements. A *package* is a database object that contains the information needed by the database manager to access data in the most efficient way for a particular application program. *Binding* is the process that creates the package the database manager needs in order to access the database when the application is executed.

Packages and cached dynamic SQL and XQuery statements can be dependent on many types of objects.

These objects could be explicitly referenced, for example, a table or user-defined function that is involved in an SQL SELECT statement. The objects could also be implicitly referenced, for example, a dependent table that needs to be checked to ensure that **referential constraints** are not violated when a row in a parent table is deleted. Packages are also dependent on the privileges which have been granted to the package creator.

If a package or cached dynamic query statement depends on an object and that object is dropped, the package or cached dynamic query statement is placed in an “invalid” state. If a package depends on a user-defined function and that function is dropped, the package is placed in an “inoperative” state, with the following conditions:

- A cached dynamic SQL or XQuery statement that is in an invalid state is automatically re-optimized on its next use. If an object required by the statement has been dropped, execution of the dynamic SQL or XQuery statement might fail with an error message.
- A package that is in an invalid state is implicitly rebound on its next use. Such a package can also be explicitly rebound. If a package was marked as being not valid because a trigger was dropped, the rebound package no longer invokes the trigger.
- A package that is in an inoperative state must be explicitly rebound before it can be used.

Federated database objects have similar dependencies. For example, dropping a server or altering a server definition invalidates any packages or cached dynamic SQL referencing nicknames associated with that server.

In some cases, it is not possible to rebind the package. For example, if a table has been dropped and not re-created, the package cannot be rebound. In this case, you must either re-create the object or change the application so it does not use the dropped object.

In many other cases, for example if one of the **constraints** was dropped, it is possible to rebind the package.

The following system catalog views help you to determine the state of a package and the package's dependencies:

- SYSCAT.PACKAGEAUTH
- SYSCAT.PACKAGEDEP
- SYSCAT.PACKAGES

Reuse of indexes with unique or primary key constraints

If you use the ALTER TABLE command to add a unique or primary key constraint to a partitioned table with a partitioned index, depending on the indexes that already exist, one might be altered to enforce the new constraint, or a new one might be created.

When you run the ALTER TABLE statement to add or change a unique or primary key for a table, a check is performed to determine whether any existing index matches the unique or primary key being defined (INCLUDE columns are ignored). An index definition matches if it identifies the same set of columns, regardless of the order or the direction (for example ASC/DESC) of the columns.

In the case of partitioned tables that have partitioned, non-unique indexes, if the index columns of the table being altered are not included among the columns that form the partition key, the index will not be considered a matching index.

If the table does have a matching index definition, it will changed to be a UNIQUE index if it wasn't one already, and will marked as required by the system. If the table has more than one existing index that matches, then an existing unique index is selected. If there is more than one matching unique index, or if there are more than one matching non-unique indexes and no matching unique indexes, then a partitioned index is favoured. Otherwise the selection of an index is arbitrary.

If no matching index is found, then a unique bidirectional index is automatically created for the columns.

Viewing constraint definitions for a table

Constraint definitions on a table can be found in the SYSCAT.INDEXES and SYSCAT.REFERENCES catalog views.

About this task

The UNIQUERULE column of the SYSCAT.INDEXES view indicates the characteristic of the index. If the value of this column is P, the index is a primary key, and if it is U, the index is a unique index (but not a primary key).

The SYSCAT.REFERENCES catalog view contains referential integrity (foreign key) constraint information.

Dropping constraints

You can explicitly drop a table check constraint using the ALTER TABLE statement, or implicitly drop it as the result of a DROP TABLE statement.

About this task

To drop constraints, use the ALTER TABLE statement with the DROP or DROP CONSTRAINT clauses. This allows you to **BIND** and continue accessing the tables that contain the affected columns. The name of all unique constraints on a table can be found in the SYSCAT.INDEXES system catalog view.

Procedure

- To explicitly drop unique constraints, use the DROP UNIQUE clause of the ALTER TABLE statement.

  The DROP UNIQUE clause of the ALTER TABLE statement drops the definition of the unique constraint *constraint-name* and all referential constraints that are dependent upon this unique constraint. The *constraint-name* must identify an existing unique constraint.

  ```
  ALTER TABLE table-name
    DROP UNIQUE constraint-name
  ```

  Dropping this unique constraint invalidates any packages or cached dynamic SQL that used the constraint.
- To drop primary key constraints, use the DROP PRIMARY KEY clause of the ALTER TABLE statement.

  The DROP PRIMARY KEY clause of the ALTER TABLE statement drops the definition of the primary key and all referential constraints that are dependent upon this primary key. The table must have a primary key. To drop a primary key using the command line, enter:

  ```
  ALTER TABLE table-name
    DROP PRIMARY KEY
  ```
- To drop (table) check constraints, use the DROP CHECK clause of the ALTER TABLE statement.

  When you drop a check constraint, all packages and cached dynamic statements with INSERT or UPDATE dependencies on the table are invalidated. The name of all check constraints on a table can be found in the SYSCAT.CHECKS catalog view. Before attempting to drop a table check constraint having a system-generated name, look for the name in the SYSCAT.CHECKS catalog view.

  The following statement drops the check constraint *constraint-name*. The *constraint-name* must identify an existing check constraint defined on the table. To drop a table check constraint using the command line:

  ```
  ALTER TABLE table_name
    DROP CHECK check_constraint_name
  ```

  Alternatively, you can use the ALTER TABLE statement with the DROP CONSTRAINT option.
- To drop foreign key (referential) constraints, use the DROP CONSTRAINT clause of the ALTER TABLE statement.

  The DROP CONSTRAINT clause of the ALTER TABLE statement drops the constraint *constraint-name*. The *constraint-name* must identify an existing foreign key constraint, primary key, or unique constraint defined on the table. To drop foreign keys using the command line, enter:

  ```
  ALTER TABLE table-name
    DROP FOREIGN KEY foreign_key_name
  ```

  When a foreign key constraint is dropped, packages or cached dynamic statements containing the following might be marked as invalid:
  - Statements that insert or update the table containing the foreign key
  - Statements that update or delete the parent table.

Example

The following examples use the DROP PRIMARY KEY and DROP FOREIGN KEY clauses in the ALTER TABLE statement to drop primary keys and foreign keys on a table:

```
ALTER TABLE EMP_ACT
  DROP PRIMARY KEY
  DROP FOREIGN KEY ACT_EMP_REF
  DROP FOREIGN KEY ACT_PROJ_REF
ALTER TABLE PROJECT
  DROP PRIMARY KEY
```

Chapter 22. Indexes

An *index* is a set of pointers that are logically ordered by the values of one or more keys. The pointers can refer to rows in a table, blocks in an MDC or ITC table, XML data in an XML storage object, and so on.

Indexes are used to:

- Improve performance. In most cases, access to data is faster with an index. Although an index cannot be created for a view, an index created for the table on which a view is based can sometimes improve the performance of operations on that view.
- Ensure uniqueness. A table with a unique index cannot have rows with identical keys.

As data is added to a table, it is appended to the bottom (unless other actions have been carried out on the table or the data being added). There is no inherent order to the data. When searching for a particular row of data, each row of the table from first to last must be checked. Indexes are used as a means to access the data within the table in an order that might otherwise not be available.

Typically, when you search for data in a table, you are looking for rows with columns that have specific values. A column value in a row of data can be used to identify the entire row. For example, an employee number would probably uniquely define a specific individual employee. Or, more than one column might be needed to identify the row. For example, a combination of customer name and telephone number. Columns in an index used to identify data rows are known as *keys*. A column can be used in more than one key.

An index is ordered by the values within a key. Keys can be unique or non-unique. Each table should have at least one unique key; but can also have other, non-unique keys. Each index has exactly one key. For example, you might use the employee ID number (unique) as the key for one index and the department number (non-unique) as the key for a different index.

Not all indexes point to rows in a table. MDC and ITC block indexes point to extents (or blocks) of the data. XML indexes for XML data use particular XML pattern expressions to index paths and values in XML documents stored within a single column. The data type of that column must be XML. Both MDC and ITC block indexes and XML indexes are system generated indexes.

Example

Table A in Figure 25 on page 320 has an index based on the employee numbers in the table. This key value provides a pointer to the rows in the table. For example, employee number 19 points to employee KMP. An index allows efficient access to rows in a table by creating a path to the data through pointers.

Unique indexes can be created to ensure uniqueness of the index key. An *index key* is a column or an ordered collection of columns on which an index is defined. Using a unique index will ensure that the value of each index key in the indexed column or columns is unique.

Figure 25 shows the relationship between an index and a table.

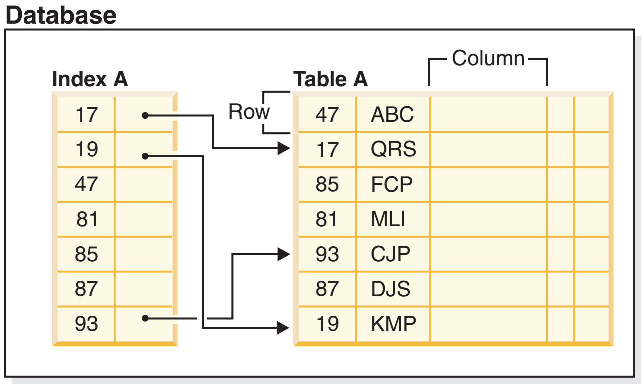


Figure 25. Relationship between an index and a table

Figure 26 illustrates the relationships among some database objects. It also shows that tables, indexes, and long data are stored in table spaces.

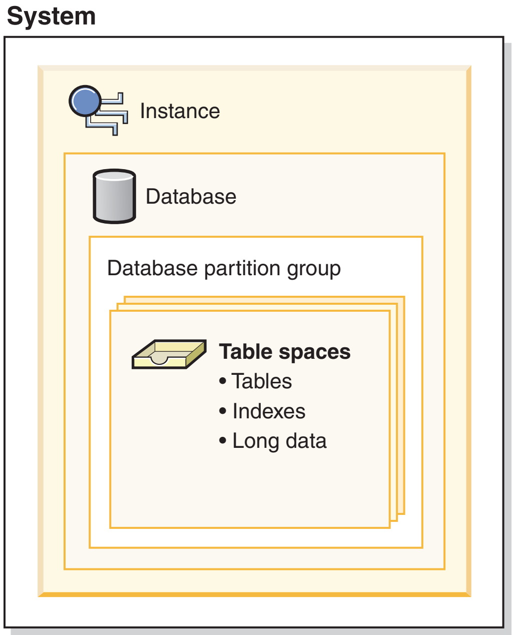


Figure 26. Relationships among selected database objects

Types of indexes

There are different types of indexes that can be created for different purposes. For example, unique indexes enforce the constraint of uniqueness in your index keys; bidirectional indexes allow for scans in both the forward and reverse directions; clustered indexes can help improve the performance of queries that traverse the table in key order.

Unique and non-unique indexes

Unique indexes are indexes that help maintain data integrity by ensuring that no two rows of data in a table have identical key values.

When attempting to create a unique index for a table that already contains data, values in the column or columns that comprise the index are checked for uniqueness; if the table contains rows with duplicate key values, the index creation process fails. Once a unique index has been defined for a table, uniqueness is enforced whenever keys are added or changed within the index. (This includes insert, update, load, import, and set integrity, to name a few.) In addition to enforcing the uniqueness of data values, a unique index can also be used to improve data retrieval performance during query processing.

Non-unique indexes, are not used to enforce constraints on the tables with which they are associated. Instead, non-unique indexes are used solely to improve query performance by maintaining a sorted order of data values that are used frequently.

Clustered and non-clustered indexes

Index architectures are classified as clustered or non-clustered. Clustered indexes are indexes whose order of the rows in the data pages correspond to the order of the rows in the index. This is why only one clustered index can exist in a given table, whereas, many non-clustered indexes can exist in the table. In some relational database management systems, the leaf node of the clustered index corresponds to the actual data, not a pointer to data that resides elsewhere.

Both clustered and non-clustered indexes contain only keys and record IDs in the index structure. The record IDs always point to rows in the data pages. The only difference between clustered and non-clustered indexes is that the database manager attempts to keep the data in the data pages in the same order as the corresponding keys appear in the index pages. Thus the database manager will attempt to insert rows with similar keys onto the same pages. If the table is reorganized, it will be inserted into the data pages in the order of the index keys.

Reorganizing a table with respect to a chosen index re-clusters the data. A clustered index is most useful for columns that have range predicates because it allows better sequential access of data in the table. This results in fewer page fetches, since like values are on the same data page.

In general, only one of the indexes in a table can have a high degree of clustering.

Clustering indexes can improve the performance of most query operations because they provide a more linear access path to data, which has been stored in pages. In addition, because rows with similar index key values are stored together, sequential detection prefetching is usually more efficient when clustering indexes are used.

However, clustering indexes cannot be specified as part of the table definition used with the CREATE TABLE statement. Instead, clustering indexes are only created by executing the CREATE INDEX statement with the CLUSTER option specified. Then the ALTER TABLE statement should be used to add a primary key that corresponds to the clustering index created to the table. This clustering index will then be used as the table's primary key index.

Note: Setting PCTFREE in the table to an appropriate value using the ALTER TABLE statement can help the table remain clustered by leaving adequate free space to insert rows in the pages with similar values. For more information, see “ALTER TABLE statement” in the *SQL Reference* and “Reducing the need to reorganize tables and indexes” in *Troubleshooting and Tuning Database Performance*.

Improving performance with clustering indexes

Generally, clustering is more effectively maintained if the clustering index is unique.

Differences between primary key or unique key constraints and unique indexes

It is important to understand that there is no significant difference between a primary unique key constraint and a unique index. The database manager uses a combination of a unique index and the NOT NULL constraint to implement the relational database concept of primary and unique key constraints. Therefore, unique indexes do not enforce primary key constraints by themselves because they allow null values. (Although null values represent unknown values, when it comes to indexing, a null value is treated as being equal to other null values.)

Therefore, if a unique index consists of a single column, only one null value is allowed-more than one null value would violate the unique constraint. Similarly, if a unique index consists of multiple columns, a specific combination of values and nulls can be used only once.

Bidirectional indexes

By default, bidirectional indexes allow scans in both the forward and reverse directions. The ALLOW REVERSE SCANS clause of the CREATE INDEX statement enables both forward and reverse index scans, that is, in the order defined at index creation time and in the opposite (or reverse) order. This option allows you to:

- Facilitate MIN and MAX functions
- Fetch previous keys
- Eliminate the need for the database manager to create a temporary table for the reverse scan
- Eliminate redundant reverse order indexes

If DISALLOW REVERSE SCANS is specified then the index cannot be scanned in reverse order. (But physically it will be exactly the same as an ALLOW REVERSE SCANS index.)

Partitioned and nonpartitioned indexes

Partitioned data can have indexes that are nonpartitioned, existing in a single table space within a database partition, indexes that are themselves partitioned across one or more table spaces within a database partition, or a combination of the two. Partitioned indexes are particularly beneficial when performing roll-in operations with partitioned tables (attaching a data partition to another table using the ATTACH PARTITION clause on the ALTER table statement.)

Indexes on partitioned tables

Partitioned tables can have indexes that are nonpartitioned (existing in a single table space within a database partition), indexes that are themselves partitioned across one or more table spaces within a database partition, or a combination of the two.

Partitioned indexes offer benefits when performing roll-in operations with partitioned tables (attaching a data partition to another table by using the ATTACH

PARTITION clause on the ALTER table statement.) With a partitioned index, you can avoid the index maintenance that you would otherwise have to perform with nonpartitioned indexes. When a partitioned table uses a nonpartitioned index, you must use the SET INTEGRITY statement to maintain the nonpartitioned index by incorporating the index keys from newly attached partitions. Not only is this time consuming, it also can require a large amount of log space, depending on the number of rows that are being rolled in.

There are some types of indexes that cannot be partitioned:

- Indexes over nonpartitioned data
- Indexes over spatial data
- XML column path indexes (system generated)

You must always create these indexes as nonpartitioned. In addition, the index key for partitioned unique indexes must include all columns from the table-partitioning key, whether they are user- or system-generated. The latter would be the case for indexes created by the system for enforcing unique or primary constraints on data.

Starting in DB2 Version 9.7 Fix Pack 1, you can create an index over XML data on a partitioned table as either partitioned or nonpartitioned. The default is a partitioned index. Unique indexes over XML data are always nonpartitioned.

Nonpartitioned indexes on partitioned tables

A *nonpartitioned index* is a single index object that refers to all rows in a partitioned table. Nonpartitioned indexes are always created as independent index objects in a single table space, even if the table data partitions span multiple table spaces.

When you create an index for a partitioned table, the index is a *partitioned index* by default unless you create one of the following types of indexes:

- A unique index where the index key does not include all of the table-partitioning columns
- A spatial index

In these cases, the index that you create is nonpartitioned. However, there are times when it is useful or necessary to create a nonpartitioned index even though your data is partitioned. In these cases, use the NOT PARTITIONED clause of the CREATE INDEX statement to create a nonpartitioned index on a partitioned table. When you create a nonpartitioned index, by default, it is stored in the same table space as the first visible or attached data partition. Figure 27 on page 324 shows an example of a single index, X1, that references all of the partitions in a table. The index was created in the same table space as the first visible partition for the table.

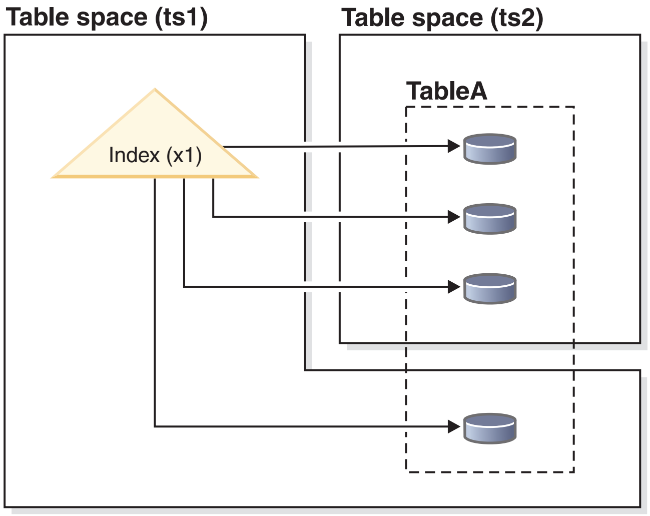


Figure 27. Nonpartitioned index on a partitioned table

Figure 28 shows an example of two nonpartitioned indexes. In this case, each index partition is in a table space separate from the table space of the data partitions. Note again how each index references all of the partitions in the table.

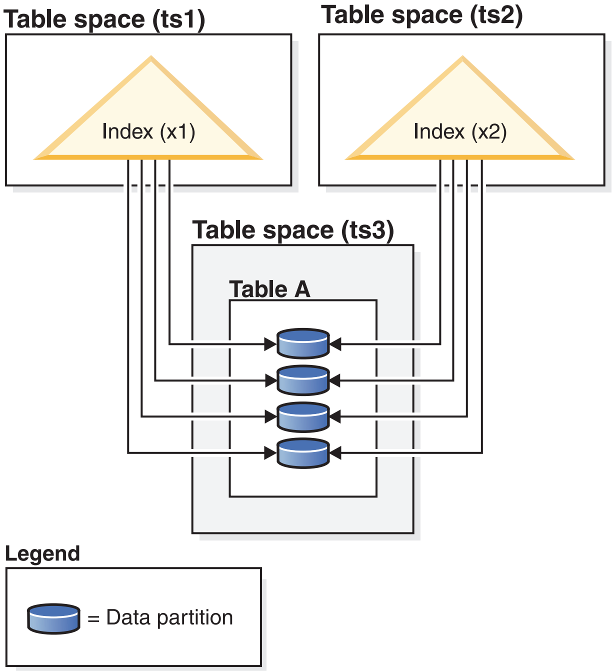


Figure 28. Nonpartitioned indexes on a partitioned table, with indexes in their own table spaces

You can override the location for a nonpartitioned index at the following times:

- When you create the table, by using the INDEX IN clause of the CREATE TABLE statement
- When you create the index, by using the IN clause of the CREATE INDEX statement.

The second approach always takes precedence over the first.

If you roll data into a partitioned table by using the ATTACH PARTITION clause of the ALTER TABLE statement, you must run the SET INTEGRITY statement to bring the attached partition data online for queries. If the indexes are nonpartitioned, bringing the attached partition data online can be a time-consuming operation that uses considerable amounts of log space, because SET INTEGRITY must insert data from the newly attached partition into the nonpartitioned indexes.

SET INTEGRITY is not required to be run after detaching a partition.

Partitioned indexes on partitioned tables

A *partitioned index* is made up of a set of *index partitions*, each of which contains the index entries for a single data partition. Each index partition contains references only to data in its corresponding data partition. Both system- and user-generated indexes can be partitioned.

A partitioned index becomes beneficial if:

- You are rolling data in or out of partitioned tables by using the ATTACH PARTITION or DETACH PARTITION clauses of the ALTER TABLE statement. With a nonpartitioned index, the SET INTEGRITY statement that you must run before the data in the newly attached partition is available can be time consuming and require large amounts of log space. When you attach a table partition that uses a partitioned index, you still must issue a SET INTEGRITY statement to perform tasks such as range validation and constraint checking.

  Tip: If data integrity checking, including range validation and other constraints checking, can be done through application logic that is independent of the data server before an attach operation, newly attached data can be made available for use much sooner. You can optimize the data roll-in process by using the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement to skip range and constraints violation checking. In this case, the table is brought out of SET INTEGRITY pending state, and the new data is available for applications to use immediately, as long as there are no nonpartitioned user indexes on the target table.

  If the indexes for the source table the index partitions for the target table, SET INTEGRITY processing does not incur the performance and log processing associated with index maintenance; the newly rolled-in data is accessible more quickly than it would be using nonpartitioned indexes. See "Conditions for matching a source table index with a target table partitioned index during ATTACH PARTITION" in *Partitioning and Clustering Guide* for more information about index matching.
- You are performing maintenance on data in a specific partition that necessitates an index reorganization. For example, consider a table with 12 partitions, each corresponding to a specific month of the year. You might want to update or delete many rows that are specific to one month of the year. This action could result in a fragmented index, which might require that you perform an index reorganization. With a partitioned index, you can reorganize just the index partition that corresponds to the data partition where the changes were made, which could save a significant amount of time compared to reorganizing an entire, nonpartitioned index.

There are some types of indexes that cannot be partitioned:

- Indexes over nonpartitioned data
- Indexes over spatial data

- XML column path indexes (system generated)

You must always create these indexes as nonpartitioned. In addition, the index key for partitioned unique indexes must include all columns from the table-partitioning key, whether they are user- or system-generated. The latter would be the case for indexes created by the system for enforcing unique or primary constraints on data.

Figure 29 shows an example of partitioned indexes.

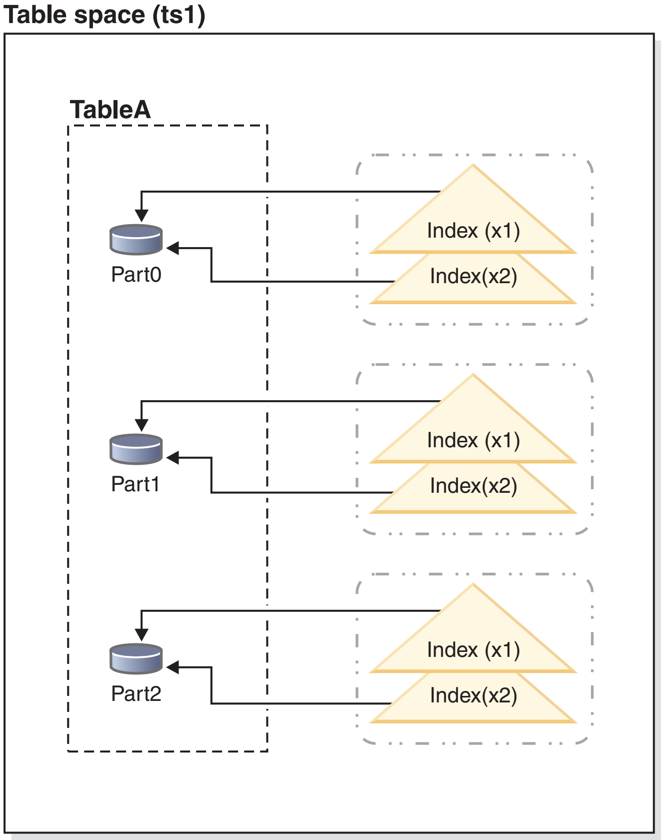


Figure 29. Partitioned indexes that share a table space with data partitions of a table

In this example, all of the data partitions for table A and all of the index partitions for table A are in a single table space. The index partitions reference only the rows in the data partition with which they are associated. (Contrast a partitioned index with a nonpartitioned index, where the index references *all* rows across *all* data partitions). Also, index partitions for a data partition are in the same index object. This particular arrangement of indexes and index partitions would be established with statements like the following statements:

```
CREATE TABLE A (columns) in ts1
      PARTITION BY RANGE (column expression)
      (PARTITION PART0 STARTING FROM constant ENDING constant,
       PARTITION PART1 STARTING FROM constant ENDING constant,
       PARTITION PART2 STARTING FROM constant ENDING constant,

   CREATE INDEX x1 ON A (...) PARTITIONED;
   CREATE INDEX x2 ON A (...) PARTITIONED;
```

Figure 30 on page 327 shows another example of a partitioned index.

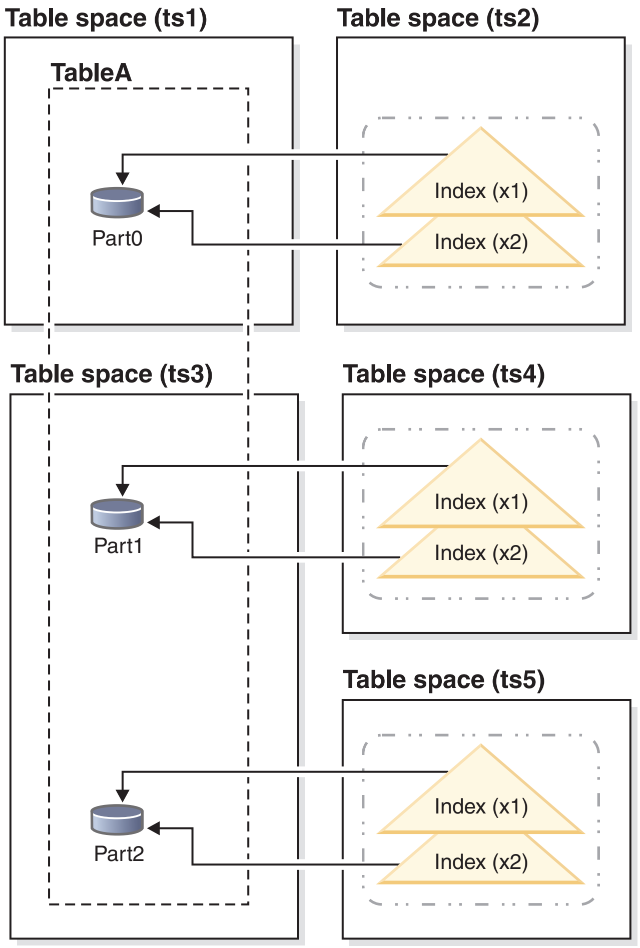


Figure 30. Partitioned indexes with data partitions and index partitions in different table spaces.

In this example, the data partitions for table A are distributed across two table spaces, TS1, and TS3. The index partitions are also in different table spaces. The index partitions reference only the rows in the data partition with which they are associated. This particular arrangement of indexes and index partitions would be established with statements like the following statements:

```
CREATE TABLE A (columns)
      PARTITION BY RANGE (column expression)
      (PARTITION PART0 STARTING FROM constant ENDING constant IN ts1 INDEX IN ts2,
       PARTITION PART1 STARTING FROM constant ENDING constant IN ts3 INDEX IN ts4,
       PARTITION PART2 STARTING FROM constant ENDING constant IN ts3,INDEX IN ts5)

   CREATE INDEX x1 ON A (...);
   CREATE INDEX x2 ON A (...);
```

In this case, the PARTITIONED clause was omitted from the CREATE INDEX statement; the indexes are still created as partitioned indexes, as this setting is the default for partitioned tables.

The Figure 31 on page 328 diagram shows an example of a partitioned table with both nonpartitioned and partitioned indexes.

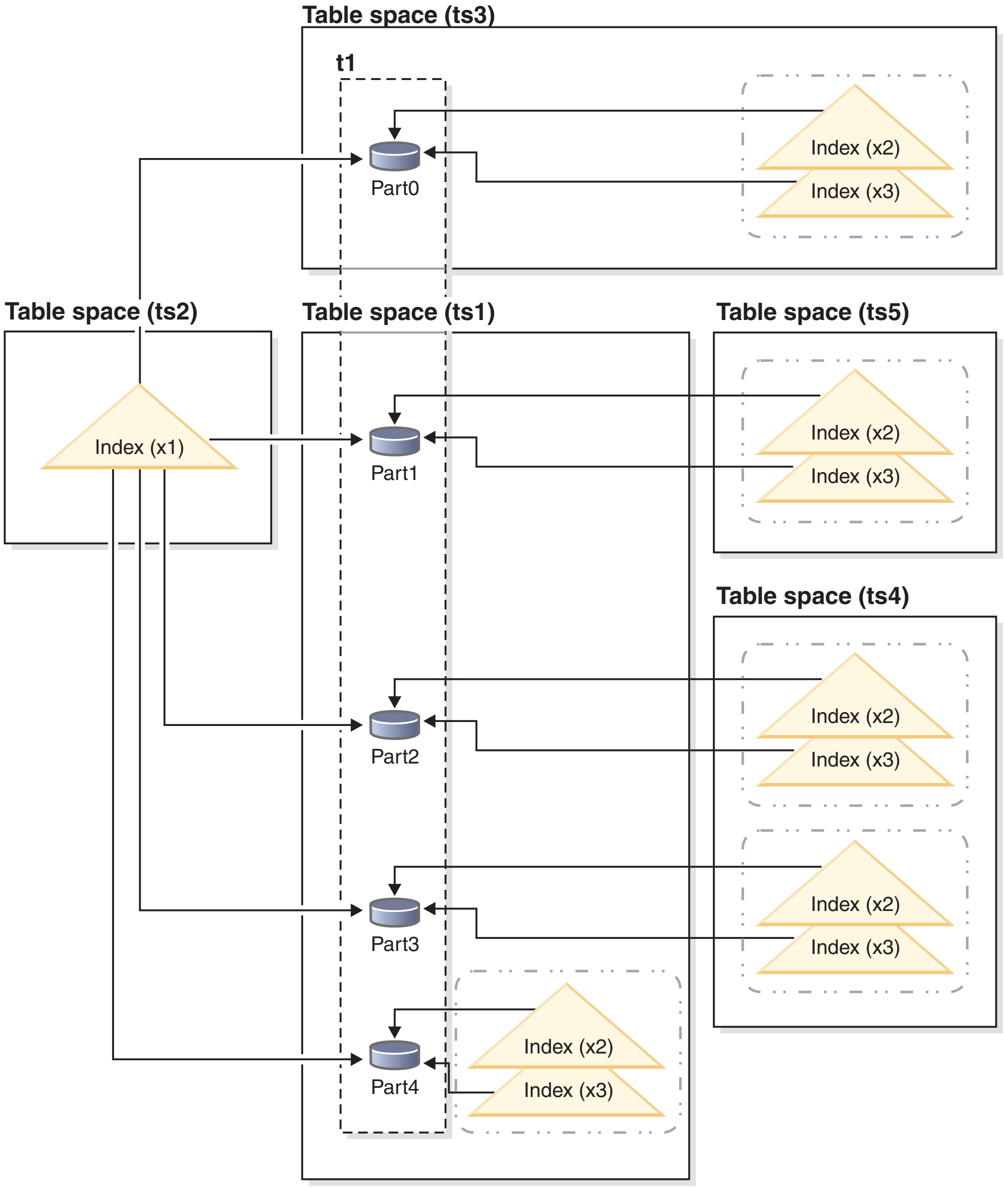


Figure 31. Combination of nonpartitioned and partitioned indexes for a partitioned table

In this diagram, index X1 is a nonpartitioned index that references all of the partitions of table T1. Indexes X2 and X3 are partitioned indexes that reside in various table spaces. This particular arrangement of indexes and index partitions would be established with statements like the following statements:

```
CREATE TABLE t1 (columns) in ts1 INDEX IN ts2 1
   PARTITION BY RANGE (column expression)
   (PARTITION PART0 STARTING FROM constant ENDING constant  IN ts3, 2
    PARTITION PART1 STARTING FROM constant ENDING constant  INDEX IN ts5,
    PARTITION PART2 STARTING FROM constant ENDING constant  INDEX IN ts4,
    PARTITION PART3 STARTING FROM constant ENDING constant  INDEX IN ts4,
    PARTITION PART4 STARTING FROM constant ENDING constant)

CREATE INDEX x1 ON t1 (...) NOT PARTITIONED;
CREATE INDEX x2 ON t1 (...) PARTITIONED;
CREATE INDEX x3 ON t1 (...) PARTITIONED;
```

Note that:

- The nonpartitioned index X1 is stored in table space TS2, because this location is the default specified (see 1) for nonpartitioned indexes for table T1.
- The index partition for data partition 0 (Part0) is stored in table space TS3, because the default location for an index partition is the same as the data partition it references (see 2).
- Part4 is stored in TS1, which is the default table space for data partitions in table T1 (see 1); the index partitions for this data partition also reside in TS1, again because the default location for an index partition is the same as the data partition it references.

Important: Unlike nonpartitioned indexes, with partitioned indexes you cannot use the INDEX IN clause of the CREATE INDEX statement to specify the table space in which to store index partitions. The only way to override the default storage location for index partitions is to specify the location at the time you create the table by using the *partition-level* INDEX IN clause of the CREATE TABLE statement. The table-level INDEX IN clause has no effect on index partition placement.

You create partitioned indexes for a partitioned table by including the PARTITIONED option in a CREATE INDEX statement. For example, for a table named SALES partitioned with `sales_date` as the table-partitioning key, to create a partitioned index, you could use a statement like this statement:

```
CREATE INDEX partIDbydate on SALES (sales_date, partID) PARTITIONED
```

If you are creating a partitioned unique index, then the table partitioning columns must be included in the index key columns. So, using the previous example, if you tried to create a partitioned index with the following statement:

```
CREATE UNIQUE INDEX uPartID on SALES (partID) PARTITIONED
```

the statement would fail because the column `sales_date`, which forms the table-partitioning key is not included in the index key.

If you omit the PARTITIONED keyword when you create an index on a partitioned table, the database manager creates a partitioned index by default unless the following conditions apply:

- You are creating a unique index, and the index key does not include all of the table-partitioning keys.
- You are creating one of the types of indexes that are described at the beginning of this topic as not able to be created as partitioned indexes.

In either of these cases, the index is created as a nonpartitioned index.

Although creating a nonpartitioned index with a definition that matches that of an existing nonpartitioned index returns SQL0605W, a partitioned index can coexist with a nonpartitioned index with a similar definition. This coexistence is intended to allow for easier adoption of partitioned indexes.

Designing indexes

Indexes are typically used to speed up access to a table. However, they can also serve a logical data design purpose.

For example, a unique index does not allow entry of duplicate values in the columns, thereby guaranteeing that no two rows of a table are the same. Indexes can also be created to order the values in a column in ascending or descending sequence.

Important: When creating indexes, keep in mind that although they can improve read performance, they negatively impact write performance. This negative impact occurs because for every row that the database manager writes to a table, it must also update any affected indexes. Therefore, create indexes only when there is a clear overall performance advantage.

When creating indexes, also take into account the structure of the tables and the type of queries that are most frequently performed on them. For example, columns that appear in the WHERE clause of a frequently issued query are good candidates for indexes. In less frequently run queries, however, the cost that an index incurs for performance in INSERT and UPDATE statements might outweigh the benefits.

Similarly, columns that figure in a GROUP BY clause of a frequent query might benefit from the creation of an index, particularly if the number of values used to group the rows is small relative to the number of rows being grouped.

When creating indexes, keep in mind that they can also be compressed. You can modify the indexes later, by enabling or disabling compression by using the ALTER INDEX statement.

To remove or delete indexes, you can use the DROP INDEX command. Dropping indexes has the reverse requirements of inserting indexes; that is, to remove (or mark as deleted) the index entries.

Guidelines and considerations when designing indexes

- Although the order of the columns that make up an index key does not make a difference to index key creation, it might make a difference to the optimizer when it is deciding whether to use an index. For example, if a query has an `ORDER BY col1,col2` clause, an index created on `(col1,col2)` could be used, but an index created on `(col2,col1)` is of no help. Similarly, if the query specified a condition such as `where col1 >= 50 and col1 <= 100` or `where col1=74`, then an index on (`col1`) or on (`col1,col2`) could be helpful, but an index on `(col2,col1)` is far less helpful.

  Note: Whenever possible, order the columns in an index key from the most distinct to the least distinct. This ordering provides the best performance.
- Any number of indexes can be defined on a particular table, to a maximum of 32 767, and they can have a beneficial effect on the performance of queries. The index manager must maintain the indexes during update, delete and insert operations. Creating a large number of indexes for a table that receives many updates can slow down processing of requests. Similarly, large index keys can also slow down processing of requests. Therefore, use indexes only where a clear advantage for frequent access exists.
- Column data which is not part of the unique index key but which is to be stored or maintained in the index is called an include column. Include columns can be specified for unique indexes only. When creating an index with include columns, only the unique key columns are sorted and considered for uniqueness. The use of include columns can enable index only access for data retrieval, thus improving performance.

- If the table that is being indexed is empty, an index is still created, but no index entries are made until the table is loaded or rows are inserted. If the table is not empty, the database manager creates the index entries while processing the CREATE INDEX statement.
- For a *clustering index*, the database manager attempts to place new rows for the table physically close to existing rows with similar key values (as defined by the index).
- If you want a *primary key index* to be a clustering index, a primary key should not be specified on the CREATE TABLE statement. Once a primary key is created, the associated index cannot be modified. Instead, issue a CREATE TABLE without a primary key clause. Then issue a CREATE INDEX statement, specifying clustering attributes. Finally, use the ALTER TABLE statement to add a primary key that corresponds to the index just created. This index is used as the primary key index.
- If you have a *partitioned table*, by default, any index that you create is a *partitioned index* unless you create a unique index that does not include the partitioning key. You can also create the index as a *nonpartitioned index*.

  Starting in DB2 Version 9.7 Fix Pack 1, you can create an index over XML data on a partitioned table as either partitioned or nonpartitioned. The default is a partitioned index.

  Partitioned indexes offer benefits when performing roll-in operations with partitioned tables (attaching a data partition to another table by using the ATTACH PARTITION clause on the ALTER table statement.) With a partitioned index, you can avoid the index maintenance that you would otherwise have to perform with nonpartitioned indexes. When a partitioned table uses a nonpartitioned index, you must use the SET INTEGRITY statement to maintain the nonpartitioned index by incorporating the index keys from newly attached partitions. Not only is this time consuming, it also can require a large amount of log space, depending on the number of rows that are being rolled in.
- Indexes consume disk space. The amount of disk space varies depending on the length of the key columns and the number of rows being indexed. The size of the index increases as more data is inserted into the table. Therefore, consider the amount of data that is being indexed when planning the size of the database. Some of the indexing sizing considerations include:
  - Primary and unique key constraints always create a system-generated unique index.
  - The creation of an MDC or ITC table also creates system-generated block indexes.
  - XML columns always cause system-generated indexes, including column path indexes and region indexes, to be created.
  - It is usually beneficial to create indexes on foreign key constraint columns.
  - Whether the index is compressed or not (by using the COMPRESS option).

Note: The maximum number of columns in an index is 64. However, if you are indexing a typed table, the maximum number of columns in an index is 63. The maximum length of an index key, including all components, is *IndexPageSize* ÷ 4. The maximum number of indexes allowed on a table is 32,767. The maximum length of an index key must not be greater than the index key length limit for the page size. For column stored lengths, see the “CREATE TABLE statement”. For the key length limits, see the “SQL and XQuery limits” topic.

- During database upgrade, existing indexes are not compressed. If a table is enabled for data row compression, new indexes created after the upgrade might be compressed, unless the COMPRESS NO option is specified on the CREATE INDEX statement.

Tools for designing indexes

Once you have created your tables, you need to consider how rapidly the database manager will be able to retrieve data from them. You can use the Design Advisor or the **db2advis** command to help you design your indexes.

Creating useful indexes on your tables can significantly improve query performance. Like indexes of a book, indexes on tables allow specific information to be located rapidly, with minimal searching. Using an index to retrieve particular rows from a table can reduce the number of expensive input/output operations that the database manager needs to perform. This is because an index allows the database manager to locate a row by reading in a relatively small number of data pages, rather than by performing an exhaustive search of all data pages until all matches are found.

The DB2 Design Advisor is a tool that can help you significantly improve your workload performance. The task of selecting which indexes, MQTs, clustering dimensions, or database partitions to create for a complex workload can be quite daunting. The Design Advisor identifies all of the objects needed to improve the performance of your workload. Given a set of SQL statements in a workload, the Design Advisor will generate recommendations for:

- New indexes
- New materialized query tables (MQTs)
- Conversion to multidimensional clustering (MDC) tables
- Redistribution of tables
- Deletion of indexes and MQTs unused by the specified workload (through the GUI tool)

You can have the Design Advisor implement some or all of these recommendations immediately or schedule them for a later time.

The Design Advisor can help simplify the following tasks:

- Planning for or setting up a new database
- Workload performance tuning

Space requirements for indexes

When designing indexes, you must be aware of their space requirements. For compressed indexes, the estimates you derive from the formulas in this topic can be used as an upper bound, however, it will likely be much smaller.

Space requirements for uncompressed indexes

For each uncompressed index, the space needed can be estimated as:

(*average index key size* + *index key overhead*) × *number of rows* × 2

where:

- The *average index key* size is the byte count of each column in the index key. When estimating the average column size for VARCHAR and VARGRAPHIC columns, use an average of the current data size, plus two bytes.

- The *index key overhead* depends on the type of table on which the index is created:

Table 73. Index key overhead for different tables

Type of table space	Table type	Index type	Index key overhead
Any	Any	XML paths or regions	11 bytes
Regular	Nonpartitioned	Any	9 bytes
	Partitioned	Partitioned	9
		Nonpartitioned	11
Large	Partitioned	Partitioned	11
		Nonpartitioned	13

- The *number of rows* is the number of rows in a table or the number of rows in a given data partition. Using the number of rows in the entire table in this calculation will give you an estimate the size for the index (for a nonpartitioned index) or for all index partitions combined (for a partitioned index). Using the number of rows in a data partition will give you an estimate of the size for the index partition.
- The factor of "2" is for overhead, such as non-leaf pages and free space.

Note:

1. For every column that allows null values, add one extra byte for the null indicator.
2. For block indexes created internally for multidimensional clustering (MDC) or insert time clustering (ITC) tables, the "number of rows" would be replaced by the "number of blocks".

Space requirements for XML indexes

For each index on an XML column, the space needed can be estimated as:

(*average index key* + *index key overhead*) × *number of indexed nodes* × 2

where:

- The *average index key* is the sum of the key parts that make up the index. The XML index is made up of several XML key parts plus a value (sql-data-type):

  14 + *variable overhead* + *byte count of sql-data-type*

  where:
  - 14 represents the number of bytes of fixed overhead
  - The *variable overhead* is the average depth of the indexed node plus 4 bytes.
  - The *byte count of sql-data-type* follows the same rules as SQL.
- The *number of indexed nodes* is the number of documents to be inserted multiplied by the number of nodes in a sample document that satisfy the XML pattern expression (XMLPATTERN) in the index definition. The *number of indexed nodes* could be the number of nodes in a partition or the entire table.

Temporary space requirements for index creation

Temporary space is required when creating the index. The maximum amount of temporary space required during index creation can be estimated as:

(*average index key size* + *index key overhead*) × *number of rows* × 3.2

For those indexes for which there could be more than one index key per row, such as spatial indexes, indexes on XML columns and internal XML regions indexes, the temporary space required can be estimated as:

(*average index key size* + *index key overhead*) × *number of indexed nodes* × 3.2

where the factor of "3.2" is for index overhead, and space required for sorting during index creation. The *number of rows* or the *number of indexed nodes* is the number in an entire table or in a given data partition.

Note: In the case of non-unique indexes, only one copy of a given duplicate key entry is stored on any given leaf node. For indexes on tables in LARGE table spaces the size for duplicate keys is 9 for nonpartitioned indexes, 7 for partitioned indexes and indexes on nonpartitioned tables. For indexes on tables in REGULAR table spaces these values are 7 for nonpartitioned indexes, 5 for partitioned indexes and indexes on nonpartitioned tables. The only exception to these rules are XML paths and XML regions indexes where the size of duplicate keys is always 7.The estimate shown previously assumes no duplicates. The space required to store an index might be over-estimated by the formula shown previously.

Temporary space is required when inserting if the number of index nodes exceeds 64 KB of data. The amount of temporary space can be estimated as:

average index key size × *number of indexed nodes* × 1.2

Estimating the number of keys per leaf page

The following two formulas can be used to estimate the number of keys per index leaf page (the second provides a more accurate estimate). The accuracy of these estimates depends largely on how well the averages reflect the actual data.

Note: For SMS table spaces, the minimum required space for leaf pages is three times the page size. For DMS table spaces, the minimum is an extent.

1. A rough estimate of the average number of keys per leaf page is:

   $((.9 * (U - (M\times2))) \times (D + 1)) \div (K + 7 + (Ds \times D))$

   where:

   - *U*, the usable space on a page, is approximately equal to the page size minus 100. For example, with a page size of 4096, U would be 3996.
   - *M* = *U* ÷ (9 + *minimumKeySize*)
   - *Ds* = *duplicateKeySize* (See the note under "Temporary space requirements for index creation".)
   - *D* = average number of duplicates per key value
   - *K* = *averageKeySize*

   Remember that *minimumKeySize* and *averageKeysize* must include an extra byte for each nullable key part, and an extra two bytes for the length of each variable length key part.

   If there are include columns, they should be accounted for in *minimumKeySize* and *averageKeySize*.

   The *minimum key size* is the sum of the key parts that make up the index:

   fixed overhead + *variable overhead* + *byte count of sql-data-type*

   where:

   - The *fixed overhead* is 13 bytes.
   - The *variable overhead* is the minimum depth of the indexed node plus 4 bytes.

- The *byte count of sql-data-type* value follows the same rules as SQL.

The `.9` can be replaced by any (100 - pctfree)/100 value, if a percent free value other than the default value of ten percent is specified during index creation.

2. A more accurate estimate of the average number of keys per leaf page is:

$$\textit{number of leaf pages} = x \;/\; (\textit{avg number of keys on leaf page})$$

where *x* is the total number of rows in the table or partition.

For the index on an XML column, *x* is the total number of indexed nodes in the column.

You can estimate the original size of an index as:

$$(L + 2L/(\textit{average number of keys on leaf page})) \times \textit{pagesize}$$

For DMS table spaces, add the sizes of all indexes on a table and round up to a multiple of the extent size for the table space on which the index resides.

You should provide additional space for index growth due to INSERT/UPDATE activity, from which page splits might result.

Use the following calculation to obtain a more accurate estimate of the original index size, as well as an estimate of the number of levels in the index. (This might be of particular interest if include columns are being used in the index definition.) The average number of keys per non-leaf page is roughly:

$$((.9 \times (U - (M \times 2))) \times (D + 1)) \div (K + 13 + (9 * D))$$

where:

- *U*, the usable space on a page, is approximately equal to the page size minus 100. For a page size of 4096, U is 3996.
- *D* is the average number of duplicates per key value on non-leaf pages (this will be much smaller than on leaf pages, and you might want to simplify the calculation by setting the value to 0).
- $M = U \div (9 + \textit{minimumKeySize}$ for non-leaf pages)
- $K = \textit{averageKeySize}$ for non-leaf pages

The *minimumKeySize* and the *averageKeySize* for non-leaf pages will be the same as for leaf pages, except when there are include columns. Include columns are not stored on non-leaf pages.

You should not replace .9 with (100 - *pctfree*)÷100, unless this value is greater than .9, because a maximum of 10 percent free space will be left on non-leaf pages during index creation.

The number of non-leaf pages can be estimated as follows:

```
if L > 1 then {P++; Z++}
While (Y > 1)
{
   P = P + Y
   Y = Y / N
  Z++
}
```

where:

- *P* is the number of pages (0 initially).
- *L* is the number of leaf pages.
- *N* is the number of keys for each non-leaf page.
- $Y = L \div N$
- *Z* is the number of levels in the index tree (1 initially).

Note: The previous calculation applies to single, nonpartitioned indexes, or to a single index partition for partitioned indexes.

Total number of pages is:

$T = (L + P + 2) \times 1.0002$

The additional 0.02% (1.0002) is for overhead, including space map pages.

The amount of space required to create the index is estimated as:

T × page size

Index compression

Indexes, including indexes on declared or created temporary tables, can be compressed in order to reduce storage costs. This is especially useful for large OLTP and data warehouse environments.

By default, index compression is enabled for compressed tables, and disabled for uncompressed tables. You can override this default behavior by using the **COMPRESS YES** option of the CREATE INDEX statement. When working with existing indexes, use the ALTER INDEX statement to enable or disable index compression; you must then perform an index reorganization to rebuild the index.

Restriction: Index compression is not supported for the following types of indexes:

- block indexes
- XML path indexes.

In addition:

- Index specifications cannot be compressed
- Compression attributes for indexes on temporary tables cannot be altered with the ALTER INDEX command.

When index compression is enabled, the on-disk and memory format of index pages are modified based on the compression algorithms chosen by the database manager so as to minimize storage space. The degree of compression achieved will vary based on the type of index you are creating, as well as the data the index contains. For example, the database manager can compress an index with a large number of duplicate keys by storing an abbreviated format of the record identifier (RID) for the duplicate keys. In an index where there is a high degree of commonality in the prefixes of the index keys, the database manager can apply compression based on the similarities in prefixes of index keys.

There can be limitations and trade-offs associated with compression. If the indexes do not share common index column values or partial common prefixes, the benefits of index compression in terms of reduced storage might be negligible. And although a unique index on a timestamp column might have very high compression capabilities due to common values for year, month, day, hour, minute, or even seconds on the same leaf page, examining if common prefixes exist could cause performance to degrade.

If you believe that compression is not offering a benefit in your particular situation, you can either re-create the indexes without compression or alter the indexes and then perform an index reorganization to disable index compression.

There are a few things you should keep in mind when you are considering using index compression:

- If you enable row compression using the **COMPRESS YES** option on the CREATE TABLE or ALTER TABLE command, then by default, compression is enabled for all indexes for which compression is supported that are created after that point for that table, unless explicitly disabled by the CREATE INDEX or ALTER INDEX commands. Similarly, if you disable row compression with the CREATE TABLE or ALTER TABLE command, index compression is disabled for all indexes created after that point for that table unless explicitly enabled by the CREATE INDEX or ALTER INDEX commands.
- If you enable index compression using the ALTER INDEX command, compression will not take place until an index reorganization is performed. Similarly, if you disable compression, the index will remain compressed until you perform an index reorganization.
- During database migration, compression is not enabled for any indexes that might have been migrated. If you want compression to be used, you must use the ALTER INDEX command and then perform an index reorganization.
- CPU usage might increase slightly as a result of the processing required for index compression or decompression. If this is not acceptable, you can disable index compression for new or existing indexes.

Examples

Example 1: Checking whether an index is compressed.

The two statements that follow create a new table T1 that is enabled for row compression, and create an index I1 on T1.

```
CREATE TABLE T1 (C1 INT, C2 INT, C3 INT) COMPRESS YES
CREATE INDEX I1 ON T1(C1)
```

By default, indexes for T1 are compressed. The *compression attribute* for index T1, which shows whether compression is enabled, can be checked by using the catalog table or the admin table function:

```
SELECT COMPRESSION FROM SYSCAT.INDEXES WHERE TABNAME='T1'

COMPRESSION
-----------
Y

   1 record(s) selected.
```

Example 2: Determining whether compressed indexes require reorganization.

To see if compressed indexes require reorganization, use the **REORGCHK** command. Figure 32 on page 338 shows the command being run on a table called T1:

```
REORGCHK ON TABLE SCHEMA1.T1

Doing RUNSTATS ....


Table statistics:

F1: 100 * OVERFLOW / CARD < 5
F2: 100 * (Effective Space Utilization of Data Pages) > 70
F3: 100 * (Required Pages / Total Pages) > 80

SCHEMA.NAME         CARD  OV   NP  FP ACTBLK   TSIZE  F1  F2  F3 REORG
----------------------------------------------------------------------
Table: SCHEMA1.T1
                     879   0   14  14      -   51861   0 100 100 ---
----------------------------------------------------------------------

Index statistics:

F4: CLUSTERRATIO or normalized CLUSTERFACTOR > 80
F5: 100 * (Space used on leaf pages / Space available on non-empty leaf pages) >
           MIN(50, (100 - PCTFREE))
F6: (100 - PCTFREE) * (Amount of space available in an index with one less level /
                       Amount of space required for all keys) < 100
F7: 100 * (Number of pseudo-deleted RIDs / Total number of RIDs) < 20
F8: 100 * (Number of pseudo-empty leaf pages / Total number of leaf pages) < 20

SCHEMA.NAME       INDCARD LEAF ELEAF LVLS NDEL KEYS LEAF_RECSIZE NLEAF_RECSIZE...
------------------------------------------------------------------------------...
Table: SCHEMA1.T1
Index: SCHEMA1.I1
                      879   15     0    2    0  682           20            20...
------------------------------------------------------------------------------...
...LEAF_PAGE_OVERHEAD NLEAF_PAGE_OVERHEAD PCT_PAGES_SAVED  F4  F5  F6  F7  F8 REORG
...--------------------------------------------------------------------------------


...               596                 596              28  56  31   -   0   0 -----
...--------------------------------------------------------------------------------
```

Figure 32. Output of **REORGCHK** *command*

The output of the **REORGCHK** command has been formatted to fit the page.

Example 3: Determining the potential space savings of index compression.

For an example of how you can calculate potential index compression savings, refer to the documentation for the ADMIN_GET_INDEX_COMPRESS_INFO table function.

Creating indexes

Indexes can be created for many reasons, including: to allow queries to run more efficiently; to order the rows of a table in ascending or descending sequence according to the values in a column; to enforce constraints such as uniqueness on index keys. You can use the CREATE INDEX statement, the DB2 Design Advisor , or the **db2advis** Design Advisor command to create the indexes.

Before you begin

On Solaris platforms, patch 122300-11 on Solaris 9 or 125100-07 on Solaris 10 is required to create indexes with RAW devices. Without this patch, the CREATE INDEX statement hangs if a RAW device is used.

About this task

This task assumes that you are creating an index on a nonpartitioned table.

Procedure

To create an index from the command line, use the CREATE INDEX statement. For example:

```
CREATE UNIQUE INDEX EMP_IX
  ON EMPLOYEE(EMPNO)
  INCLUDE(FIRSTNAME, JOB)
```

The INCLUDE clause, applicable only on unique indexes, specifies additional columns to be appended to the set of index key columns. Any columns included with this clause are not used to enforce uniqueness. These included columns can improve the performance of some queries through index only access. This option might:

- Eliminate the need to access data pages for more queries
- Eliminate redundant indexes

If SELECT EMPNO, FIRSTNAME, JOB FROM EMPLOYEE is issued to the table on which this index resides, all of the required data can be retrieved from the index without reading data pages. This improves performance.

What to do next

When a row is deleted or updated, the index keys are marked as deleted and are not physically removed from a page until cleanup is done some time after the deletion or update is committed. These keys are referred to as pseudo-deleted keys. Such a cleanup might be done by a subsequent transaction which is changing the page where the key is marked deleted. Clean up of pseudo-deleted keys can be explicitly triggered by using the **CLEANUP ONLY ALL** parameter in the **REORG INDEXES** command.

Creating nonpartitioned indexes on partitioned tables

When you create a *nonpartitioned index* on a partitioned table, you create a single index object that refers to all rows in the table. Nonpartitioned indexes are always created in a single table space, even if the table data partitions span multiple table spaces.

Before you begin

This task assumes that your partitioned table has already been created.

Procedure

1. Formulate a CREATE INDEX statement for your table, using the NOT PARTITIONED clause. For example:

   ```
   CREATE INDEX indexName ON tableName(column) NOT PARTITIONED
   ```
2. Execute the CREATE INDEX statement from a supported DB2 interface.

Example

Example 1: Creating a nonpartitioned index in the same table space as the data partition.

Assume the SALES table is defined as follows:

```
CREATE TABLE sales(store_num INT, sales_date DATE, total_sales DECIMAL (6,2)) IN ts1
   PARTITION BY RANGE(store_num)
   (STARTING FROM (1) ENDING AT (100),
    STARTING FROM (101) ENDING AT (150),
    STARTING FROM (151) ENDING AT (200))
```

The three partitions of the SALES table are stored in table space TS1. By default, any indexes created for this table are also stored in TS1, because that was the table space specified for this table. To create a nonpartitioned index STORENUM on the STORE_NUM column, use the following statement:

```
CREATE INDEX StoreNum ON sales(store_num) NOT PARTITIONED
```

Note that the NOT PARTITIONED clause is required, otherwise the index is created as a partitioned index, the default for partitioned tables.

Example 2: Creating a nonpartitioned index in a table space other than the default

Assume that a table called PARTS is defined as follows:

```
CREATE TABLE parts(part_number INT, manufacturer CHAR, description CLOB,
   price DECIMAL (4,2)) IN ts1 INDEX in ts2
   PARTITION BY RANGE (part_number)
   (STARTING FROM (1) ENDING AT (10) IN ts3,
    STARTING FROM (11) ENDING AT (20) INDEX IN ts1,
    STARTING FROM (21) ENDING AT (30) IN ts2 INDEX IN ts4);
```

The PARTS table consists of three partitions: the first is in table space TS3, the second is in TS2 and the third in TS3. If you issue the following statement a nonpartitioned index that orders the rows in descending order of manufacturer name is created:

```
CREATE INDEX manufct on parts(manufacturer DESC) NOT PARTITIONED IN TS3;
```

This index is created in table space TS3; the INDEX IN clause of the CREATE TABLE statement is overridden by the IN *tablespace* clause of the CREATE INDEX statement. Because the table PARTS is partitioned, you must include the NOT PARTITIONED clause in the CREATE INDEX statement to create a nonpartitioned index.

Creating partitioned indexes

When you create a *partitioned index* for a partitioned table, each data partition is indexed in its own index partition. By default, the index partition is stored in same table space as the data partition it indexes. Data in the indexes is distributed based on the distribution key of the table.

Before you begin

This task assumes that your partitioned table has already been created.

About this task

Restrictions

There are some types of indexes that cannot be partitioned:

- Indexes over nonpartitioned data
- Indexes over spatial data
- XML column path indexes (system generated)

You must always create these indexes as nonpartitioned. In addition, the index key for partitioned unique indexes must include all columns from the table-partitioning key, whether they are user- or system-generated. The latter would be the case for indexes created by the system for enforcing unique or primary constraints on data.

Also, The IN clause of the CREATE INDEX statement is not supported for creating partitioned indexes. By default, index partitions are created in the same table space as the data partitions they index. To specify an alternative table space in which to store the index partition, you must use the partition-level INDEX IN clause of the CREATE TABLE statement to specify a table space for indexes on a partition-by-partition basis. If you omit this clause, the index partitions will reside in the same table space as the data partitions they index.

Procedure

1. Formulate a CREATE INDEX statement for your table, using the PARTITIONED clause.
2. Execute the CREATE INDEX statement from a supported DB2 interface.

Example

Note: These examples are for illustrative purposes only, and do not reflect best practices for creating partitioned tables or indexes.
Example 1: Creating a partitioned index in the same table spaces as the data partition.

In this example, assume the SALES table is has been defined as follows:

```
CREATE TABLE sales(store_num INT, sales_date DATE, total_sales DECIMAL (6,2))
 IN ts1
 PARTITION BY RANGE(store_num)
   (STARTING FROM (1) ENDING AT (100),
    STARTING FROM (101) ENDING AT (150),
    STARTING FROM (151) ENDING AT (200))
```

In this case, the three partitions of the table SALES are stored in table space `ts1`. Any partitioned indexes created for this table will also be stored in `ts1`, because that is the table space in which each partition for this table will be stored. To create a partitioned index on the store number, use the following statement:

```
CREATE INDEX StoreNum ON sales(store_num) PARTITIONED
```

Example 2: Choosing an alternative location for all index partitions.

In this example, assume the EMPLOYEE table is has been defined as follows:

```
CREATE TABLE employee(employee_number INT, employee_name CHAR,
              job_code INT, city CHAR, salary DECIMAL (6,2))
              IN ts1 INDEX IN ts2
   PARTITION BY RANGE (job_code)
   (STARTING FROM (1) ENDING AT (10) INDEX IN ts2,
    STARTING FROM (11) ENDING AT (20) INDEX IN ts2,
    STARTING FROM (21) ENDING AT (30) INDEX IN ts2)
```

To create a partitioned index on the job codes, use the following statement:

```
CREATE INDEX JobCode ON employee(job_code) PARTITIONED
```

In this example, the partitions of the EMPLOYEE table are stored in table space `ts1`, however, all index partitions will be stored in `ts2`.

Example 3: Indexes created in several partitions.

Assume a table called PARTS has been defined as follows:

```
CREATE TABLE parts(part_number INT, manufacturer CHAR,
             description CLOB, price DECIMAL (4,2)) IN ts1 INDEX in ts2
   PARTITION BY RANGE (part_number)
   (STARTING FROM (1) ENDING AT (10) IN ts3,
    STARTING FROM (11) ENDING AT (20) INDEX IN ts1,
    STARTING FROM (21) ENDING AT (30) IN ts2 INDEX IN ts4);
```

In this case, the PARTS table consists of three partitions: the first is in table space ts3, the second in ts1 and the 3rd in ts2. If the following statements are issued:

```
CREATE INDEX partNoasc ON parts(part_number ASC)  PARTITIONED
CREATE INDEX manufct on parts(manufacturer DESC) NOT PARTITIONED IN TS3;
```

then two indexes are created. The first is a partitioned index to order the rows in ascending order of part number. The first index partition is created in table space ts3, the second in ts1 and the third in ts4. The second index is a nonpartitioned index which orders the rows in descending order of the manufacturer's name. This index is created in ts3. Note that the IN clause is allowed in CREATE INDEX statements for nonpartitioned indexes. Also, in this case, because the table PARTS is partitioned, to create a nonpartitioned index, the clause NOT PARTITIONED must be included in the CREATE INDEX statement.

Modifying indexes

If you want to modify your index, other than using the ALTER INDEX statement to enable or disable index compression, you must drop the index first and then create the index again.

Example

For example, you cannot add a column to the list of key columns without dropping the previous definition and creating a new index. You can, however, add a comment to describe the purpose of the index using the COMMENT statement.

Renaming indexes

You can use the RENAME statement to rename an existing index.

About this task

When renaming an index, the source index must not be a system-generated index.

Procedure

To rename an existing index, issue the following statement from the command line:

```
RENAME INDEX source_index_name TO target_index_name
```

source_index_name is the name of the existing index that is to be renamed. The name, including the schema name, must identify an index that exists in the database. It must not be the name of an index on a declared temporary table or on a created temporary table. The schema name must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT.
target_index_name specifies the new name for the index without a schema name. The schema name of the source object is used to qualify the new name for the object. The qualified name must not identify an index that exists in the database.

Results

If the RENAME statement is successful, the system catalog tables are updated to reflect the new index name.

Rebuilding indexes

Certain database operations, such as a rollforward through a create index that was not fully logged, can cause an index object to become invalid because the index is not created during the rollforward operation. The index object can be recovered by recreating the indexes in it.

About this task

When the database manager detects that an index is no longer valid, it automatically attempts to rebuild it. When the rebuild takes place, it is controlled by the **indexrec** parameter of the database or database manager configuration file. There are five possible settings for this:

- SYSTEM
- RESTART
- RESTART_NO_REDO
- ACCESS
- ACCESS_NO_REDO

RESTART_NO_REDO and ACCESS_NO_REDO are similar to RESTART and ACCESS.

The NO_REDO options mean that even if the index was fully logged during the original operation, such as CREATE INDEX, the index will not be recreated during rollforward, but will instead be created either at restart time or first access. See the **indexrec** parameter for more information.

If database restart time is not a concern, it is better for invalid indexes to be rebuilt as part of the process of returning a database to a consistent state. When this approach is used, the time needed to restart a database will be longer due to the index recreation process; however, normal processing will not be impacted once the database has been returned to a consistent state.

On the other hand, when indexes are rebuilt as they are accessed, the time taken to restart a database is faster, but an unexpected degradation in response time can occur as a result of an index being recreated; for example, users accessing a table that has an invalid index would have to wait for the index to be rebuilt. In addition, unexpected locks can be acquired and held long after an invalid index has been recreated, especially if the transaction that caused the index recreation to occur never terminates (that is, commits or rolls back the changes made).

Dropping indexes

To delete an index, use the DROP statement.

About this task

Other than changing the COMPRESSION attribute of an index, you cannot change any clause of an index definition; you must drop the index and create it again. Dropping an index does not cause any other objects to be dropped but might cause some packages to be invalidated.

Restrictions

A primary key or unique key index cannot be explicitly dropped. You must use one of the following methods to drop it:

- If the primary index or unique constraint was created automatically for the primary key or unique key, dropping the primary key or unique key causes the index to be dropped. Dropping is done through the ALTER TABLE statement.
- If the primary index or the unique constraint was user-defined, the primary key or unique key must be dropped first, through the ALTER TABLE statement. After the primary key or unique key is dropped, the index is no longer considered the primary index or unique index, and it can be explicitly dropped.

Procedure

To drop an index by using the command line, enter:

```
DROP INDEX index_name
```

Results

Any packages and cached dynamic SQL and XQuery statements that depend on the dropped indexes are marked invalid. The application program is not affected by changes resulting from adding or dropping indexes.

Chapter 23. Triggers

A *trigger* defines a set of actions that are performed in response to an insert, update, or delete operation on a specified table. When such an SQL operation is executed, the trigger is said to have been *activated*. Triggers are optional and are defined using the CREATE TRIGGER statement.

Triggers can be used, along with referential constraints and check constraints, to enforce data integrity rules. Triggers can also be used to cause updates to other tables, automatically generate or transform values for inserted or updated rows, or invoke functions to perform tasks such as issuing alerts.

Triggers are a useful mechanism for defining and enforcing *transitional* business rules, which are rules that involve different states of the data (for example, a salary that cannot be increased by more than 10 percent).

Using triggers places the logic that enforces business rules inside the database. This means that applications are not responsible for enforcing these rules. Centralized logic that is enforced on all of the tables means easier maintenance, because changes to application programs are not required when the logic changes.

The following are specified when creating a trigger:

- The *subject table* specifies the table for which the trigger is defined.
- The *trigger event* defines a specific SQL operation that modifies the subject table. The event can be an insert, update, or delete operation.
- The *trigger activation time* specifies whether the trigger should be activated before or after the trigger event occurs.

The statement that causes a trigger to be activated includes a *set of affected rows*. These are the rows of the subject table that are being inserted, updated, or deleted. The *trigger granularity* specifies whether the actions of the trigger are performed once for the statement or once for each of the affected rows.

The *triggered action* consists of an optional search condition and a set of statements that are executed whenever the trigger is activated. The statements are only executed if the search condition evaluates to true. If the trigger activation time is before the trigger event, triggered actions can include statements that select, set transition variables, or signal SQL states. If the trigger activation time is after the trigger event, triggered actions can include statements that select, insert, update, delete, or signal SQL states.

The triggered action can refer to the values in the set of affected rows using *transition variables*. Transition variables use the names of the columns in the subject table, qualified by a specified name that identifies whether the reference is to the old value (before the update) or the new value (after the update). The new value can also be changed using the SET Variable statement in before, insert, or update triggers.

Another means of referring to the values in the set of affected rows is to use *transition tables*. Transition tables also use the names of the columns in the subject table, but specify a name to allow the complete set of affected rows to be treated as a table. Transition tables can only be used in AFTER triggers (that is, not with BEFORE and INSTEAD OF triggers), and separate transition tables can be defined for old and new values.

Multiple triggers can be specified for a combination of table, event (INSERT, UPDATE, DELETE), or activation time (BEFORE, AFTER, INSTEAD OF). When more than one trigger exists for a particular table, event, and activation time, the order in which the triggers are activated is the same as the order in which they were created. Thus, the most recently created trigger is the last trigger to be activated.

The activation of a trigger might cause *trigger cascading*, which is the result of the activation of one trigger that executes statements that cause the activation of other triggers or even the same trigger again. The triggered actions might also cause updates resulting from the application of referential integrity rules for deletions that can, in turn, result in the activation of additional triggers. With trigger cascading, a chain of triggers and referential integrity delete rules can be activated, causing significant change to the database as a result of a single INSERT, UPDATE, or DELETE statement.

When multiple triggers have insert, update, or delete actions against the same object, conflict resolution mechanism, like temporary tables, are used to resolve access conflicts, and this can have a noticeable impact on performance, particularly in partitioned database environments.

Types of triggers

A *trigger* defines a set of actions that are performed in response to an insert, update, or delete operation on a specified table. When such an SQL operation is executed, the trigger is said to have been *activated*. Triggers are optional and are defined using the CREATE TRIGGER statement.

Triggers can be used, along with referential constraints and check constraints, to enforce data integrity rules. Triggers can also be used to cause updates to other tables, automatically generate or transform values for inserted or updated rows, or invoke functions to perform tasks such as issuing alerts.

The following types of triggers are supported:

BEFORE triggers

Run before an update, or insert. Values that are being updated or inserted can be modified before the database is actually modified. You can use triggers that run before an update or insert in several ways:

- To check or modify values before they are actually updated or inserted in the database. This is useful if you must transform data from the way the user sees it to some internal database format.
- To run other non-database operations coded in user-defined functions.

BEFORE DELETE triggers

Run before a delete. Checks values (a raises an error, if necessary).

AFTER triggers

Run after an update, insert, or delete. You can use triggers that run after an update or insert in several ways:

- To update data in other tables. This capability is useful for maintaining relationships between data or in keeping audit trail information.

- To check against other data in the table or in other tables. This capability is useful to ensure data integrity when referential integrity constraints aren't appropriate, or when table check constraints limit checking to the current table only.
- To run non-database operations coded in user-defined functions. This capability is useful when issuing alerts or to update information outside the database.

INSTEAD OF triggers
Describe how to perform insert, update, and delete operations against views that are too complex to support these operations natively. They allow applications to use a view as the sole interface for all SQL operations (insert, delete, update and select).

BEFORE triggers

By using triggers that run before an update or insert, values that are being updated or inserted can be modified before the database is actually modified. These can be used to transform input from the application (user view of the data) to an internal database format where desired.

These BEFORE triggers can also be used to cause other non-database operations to be activated through user-defined functions.

BEFORE DELETE triggers run before a delete operation. They check the values and raise an error, if necessary.

Examples

The following example defines a DELETE TRIGGER with a complex default:

```
CREATE TRIGGER trigger1
  BEFORE UPDATE ON table1
  REFERENCING NEW AS N
  WHEN (N.expected_delivery_date IS NULL)
  SET N.expected_delivery_date = N.order_date + 5 days;
```

The following example defines a DELETE TRIGGER with a cross table constraint that is not a referential integrity constraint:

```
CREATE TRIGGER trigger2
  BEFORE UPDATE ON table2
  REFERENCING NEW AS N
  WHEN (n.salary > (SELECT maxsalary FROM salaryguide WHERE rank = n.position))
  SIGNAL SQLSTATE '78000' SET MESSAGE_TEXT = 'Salary out of range');
```

AFTER triggers

Triggers that run after an update, insert, or delete can be used in several ways.

- Triggers can update, insert, or delete data in the same or other tables. This is useful to maintain relationships between data or to keep audit trail information.
- Triggers can check data against values of data in the rest of the table or in other tables. This is useful when you cannot use referential integrity constraints or check constraints because of references to data from other rows from this or other tables.
- Triggers can use user-defined functions to activate non-database operations. This is useful, for example, for issuing alerts or updating information outside the database.

Example

The following example presents an AFTER trigger that increases the number of employees when a new employee is hired.

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

INSTEAD OF triggers

INSTEAD OF triggers describe how to perform insert, update, and delete operations against complex views. INSTEAD OF triggers allow applications to use a view as the sole interface for all SQL operations (insert, delete, update and select).

Usually, INSTEAD OF triggers contain the inverse of the logic applied in a view body. For example, consider a view that decrypts columns from its source table. The INSTEAD OF trigger for this view encrypts data and then inserts it into the source table, thus performing the symmetrical operation.

Using an INSTEAD OF trigger, the requested modify operation against the view gets replaced by the trigger logic, which performs the operation on behalf of the view. From the perspective of the application this happens transparently, as it perceives that all operations are performed against the view. Only one INSTEAD OF trigger is allowed for each kind of operation on a given subject view.

The view itself must be an untyped view or an alias that resolves to an untyped view. Also, it cannot be a view that is defined using WITH CHECK OPTION (a symmetric view) or a view on which a symmetric view has been defined directly or indirectly.

Example

The following example presents three INSTEAD OF triggers that provide logic for INSERTs, UPDATEs, and DELETEs to the defined view (EMPV). The view EMPV contains a join in its from clause and therefore cannot natively support any modify operation.

```
CREATE VIEW EMPV(EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO,
                 HIREDATE, DEPTNAME)
AS SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO,
          HIREDATE, DEPTNAME
          FROM EMPLOYEE, DEPARTMENT WHERE
               EMPLOYEE.WORKDEPT = DEPARTMENT.DEPTNO

CREATE TRIGGER EMPV_INSERT INSTEAD OF INSERT ON EMPV
REFERENCING NEW AS NEWEMP FOR EACH ROW
INSERT INTO EMPLOYEE (EMPNO, FIRSTNME, MIDINIT, LASTNAME,
                      WORKDEPT, PHONENO, HIREDATE)
        VALUES(EMPNO, FIRSTNME, MIDINIT, LASTNAME,
               COALESCE((SELECT DEPTNO FROM DEPARTMENT AS D
                        WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                        RAISE_ERROR('70001', 'Unknown dept name')),
               PHONENO, HIREDATE)

CREATE TRIGGER EMPV_UPDATE INSTEAD OF UPDATE ON EMPV
REFERENCING NEW AS NEWEMP OLD AS OLDEMP
  FOR EACH ROW
BEGIN ATOMIC
 VALUES(CASE WHEN NEWEMP.EMPNO = OLDEMP.EMPNO THEN 0
             ELSE RAISE_ERROR('70002', 'Must not change EMPNO') END);
 UPDATE EMPLOYEE AS E
   SET (FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE)
     = (NEWEMP.FIRSTNME, NEWEMP.MIDINIT, NEWEMP.LASTNAME,
               COALESCE((SELECT DEPTNO FROM DEPARTMENT AS D
                         WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                        RAISE_ERROR ('70001', 'Unknown dept name')),
               NEWEMP.PHONENO, NEWEMP.HIREDATE)
 WHERE NEWEMP.EMPNO = E.EMPNO;
END

CREATE TRIGGER EMPV_DELETE INSTEAD OF DELETE ON EMPV
REFERENCING OLD AS OLDEMP FOR EACH ROW
DELETE FROM EMPLOYEE AS E WHERE E.EMPNO = OLDEMP.EMPNO
```

Designing triggers

When creating a trigger, you must associate it with a table; when creating an INSTEAD OF trigger, you must associate it with a view. This table or view is called the *target table* of the trigger. The term modify operation refers to any change in the state of the target table.

About this task

A *modify operation* is initiated by:
- an INSERT statement
- an UPDATE statement, or a referential constraint which performs an UPDATE
- a DELETE statement, or a referential constraint which performs a DELETE
- a MERGE statement

You must associate each trigger with one of these three types of modify operations. The association is called the *trigger event* for that particular trigger.

You must also define the action, called the *triggered action*, that the trigger performs when its trigger event occurs. The triggered action consists of one or more statements which can execute either before or after the database manager performs the trigger event. Once a trigger event occurs, the database manager determines the set of rows in the subject table that the modify operation affects and executes the trigger.

Guidelines when creating triggers:

When creating a trigger, you must declare the following attributes and behavior:
- The name of the trigger.
- The name of the subject table.
- The trigger activation time (BEFORE or AFTER the modify operation executes).
- The trigger event (INSERT, DELETE, or UPDATE).
- The old transition variable value, if any.
- The new transition variable value, if any.
- The old transition table value, if any.
- The new transition table value, if any.
- The granularity (FOR EACH STATEMENT or FOR EACH ROW).
- The triggered action of the trigger (including a triggered action condition and triggered statement(s)).

- If the trigger event is UPDATE a trigger-column list if the trigger should only fire when specific columns are specified in the update statement.

Designing multiple triggers:

When triggers are defined using the CREATE TRIGGER statement, their creation time is registered in the database in form of a timestamp. The value of this timestamp is subsequently used to order the activation of triggers when there is more than one trigger that should be run at the same time. For example, the timestamp is used when there is more than one trigger on the same subject table with the same event and the same activation time. The timestamp is also used when there are one or more AFTER or INSTEAD OF triggers that are activated by the trigger event and referential constraint actions caused directly or indirectly (that is, recursively by other referential constraints) by the triggered action.

Consider the following two triggers:

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  BEGIN ATOMIC
    UPDATE COMPANY_STATS
    SET NBEMP = NBEMP + 1;
  END

CREATE TRIGGER NEW_HIRED_DEPT
   AFTER INSERT ON EMPLOYEE
   REFERENCING NEW AS EMP
   FOR EACH ROW
     BEGIN ATOMIC
       UPDATE DEPTS
       SET NBEMP = NBEMP + 1
       WHERE DEPT_ID = EMP.DEPT_ID;
     END
```

The preceding triggers are activated when you run an INSERT operation on the employee table. In this case, the timestamp of their creation defines which of the preceding two triggers is activated first.

The activation of the triggers is conducted in ascending order of the timestamp value. Thus, a trigger that is newly added to a database runs after all the other triggers that are previously defined.

Old triggers are activated before new triggers to ensure that new triggers can be used as *incremental* additions to the changes that affect the database. For example, if a triggered statement of trigger T1 inserts a new row into a table T, a triggered statement of trigger T2 that is run after T1 can be used to update the same row in T with specific values. Because the activation order of triggers is predictable, you can have multiple triggers on a table and still know that the newer ones will be acting on a table that has already been modified by the older ones.

Trigger interactions with referential constraints:

A trigger event can occur as a result of changes due to referential constraint enforcement. For example, given two tables DEPT and EMP, if deleting or updating DEPT causes propagated deletes or updates to EMP by means of referential integrity constraints, then delete or update triggers defined on EMP become activated as a result of the referential constraint defined on DEPT. The triggers on EMP are run either BEFORE or AFTER the deletion (in the case of ON DELETE CASCADE) or update of rows in EMP (in the case of ON DELETE SET NULL), depending on their activation time.

Specifying what makes a trigger fire (triggering statement or event)

Every trigger is associated with an event. Triggers are activated when their corresponding event occurs in the database. This trigger event occurs when the specified action, either an UPDATE, INSERT, or DELETE statement (including those caused by actions of referential constraints), is performed on the target table.

About this task

For example:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

The preceding statement defines the trigger new_hire, which activates when you perform an insert operation on table employee.

You associate every trigger event, and consequently every trigger, with exactly one target table and exactly one modify operation. The modify operations are:

Insert operation
An insert operation can only be caused by an INSERT statement or the insert operation of a MERGE statement. Therefore, triggers are not activated when data is loaded using utilities that do not use INSERT, such as the **LOAD** command.

Delete operation
A delete operation can be caused by a DELETE statement, or the delete operation of a MERGE statement, or as a result of a referential constraint rule of ON DELETE CASCADE.

Update operation
An update operation can be caused by an UPDATE statement, or the update operation of a MERGE statement, or as a result of a referential constraint rule of ON DELETE SET NULL.

If the trigger event is an update operation, the event can be associated with specific columns of the target table. In this case, the trigger is only activated if the update operation attempts to update any of the specified columns. This provides a further refinement of the event that activates the trigger.

For example, the following trigger, REORDER, activates only if you perform an update operation on the columns ON_HAND or MAX_STOCKED, of the table PARTS:

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
  FOR EACH ROW
  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
  BEGIN ATOMIC
  VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                              N_ROW.ON_HAND,
                              N_ROW.PARTNO));
  END
```

When a trigger is activated, it runs according to its level of granularity as follows:

FOR EACH ROW

It runs as many times as the number of rows in the set of affected rows. If you need to refer to the specific rows affected by the triggered action, use FOR EACH ROW granularity. An example of this is the comparison of the new and old values of an updated row in an AFTER UPDATE trigger.

FOR EACH STATEMENT

It runs once for the entire trigger event.

If the set of affected rows is empty (that is, in the case of a searched UPDATE or DELETE in which the WHERE clause did not qualify any rows), a FOR EACH ROW trigger does not run. But a FOR EACH STATEMENT trigger still runs once.

For example, keeping a count of number of employees can be done using FOR EACH ROW.

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

You can achieve the same affect with one update by using a granularity of FOR EACH STATEMENT.

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  REFERENCING NEW_TABLE AS NEWEMPS
  FOR EACH STATEMENT
  UPDATE COMPANY_STATS
  SET NBEMP = NBEMP + (SELECT COUNT(*) FROM NEWEMPS)
```

Note:

- A granularity of FOR EACH STATEMENT is not supported for BEFORE triggers.
- The maximum nesting level for triggers is 16. That is, the maximum number of cascading trigger activations is 16. A trigger activation refers to the activation of a trigger upon a triggering event, such as insert, update, or delete of data in a column of a table, or generally to a table.

Specifying when a trigger fires (BEFORE, AFTER, and INSTEAD OF clauses)

The *trigger activation time* specifies when the trigger should be activated, relative to the trigger event.

About this task

There are three activation times that you can specify: BEFORE, AFTER, or INSTEAD OF:

- If the activation time is BEFORE, the triggered actions are activated for each row in the set of affected rows before the trigger event executes. Hence, the subject table will only be modified after the BEFORE trigger has completed execution for each row. Note that BEFORE triggers must have a granularity of FOR EACH ROW.
- If the activation time is AFTER, the triggered actions are activated for each row in the set of affected rows or for the statement, depending on the trigger granularity. This occurs after the trigger event has been completed, and after the database manager checks all constraints that the trigger event might affect,

including actions of referential constraints. Note that AFTER triggers can have a granularity of either FOR EACH ROW or FOR EACH STATEMENT.

For example, the activation time of the following trigger is AFTER the INSERT operation on `employee`:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

- If the activation time is INSTEAD OF, the triggered actions are activated for each row in the set of affected rows instead of executing the trigger event. INSTEAD OF triggers must have a granularity of FOR EACH ROW, and the subject table must be a view. No other triggers are able to use a view as the subject table.

Example

The following diagram illustrates the execution model of BEFORE and AFTER triggers:

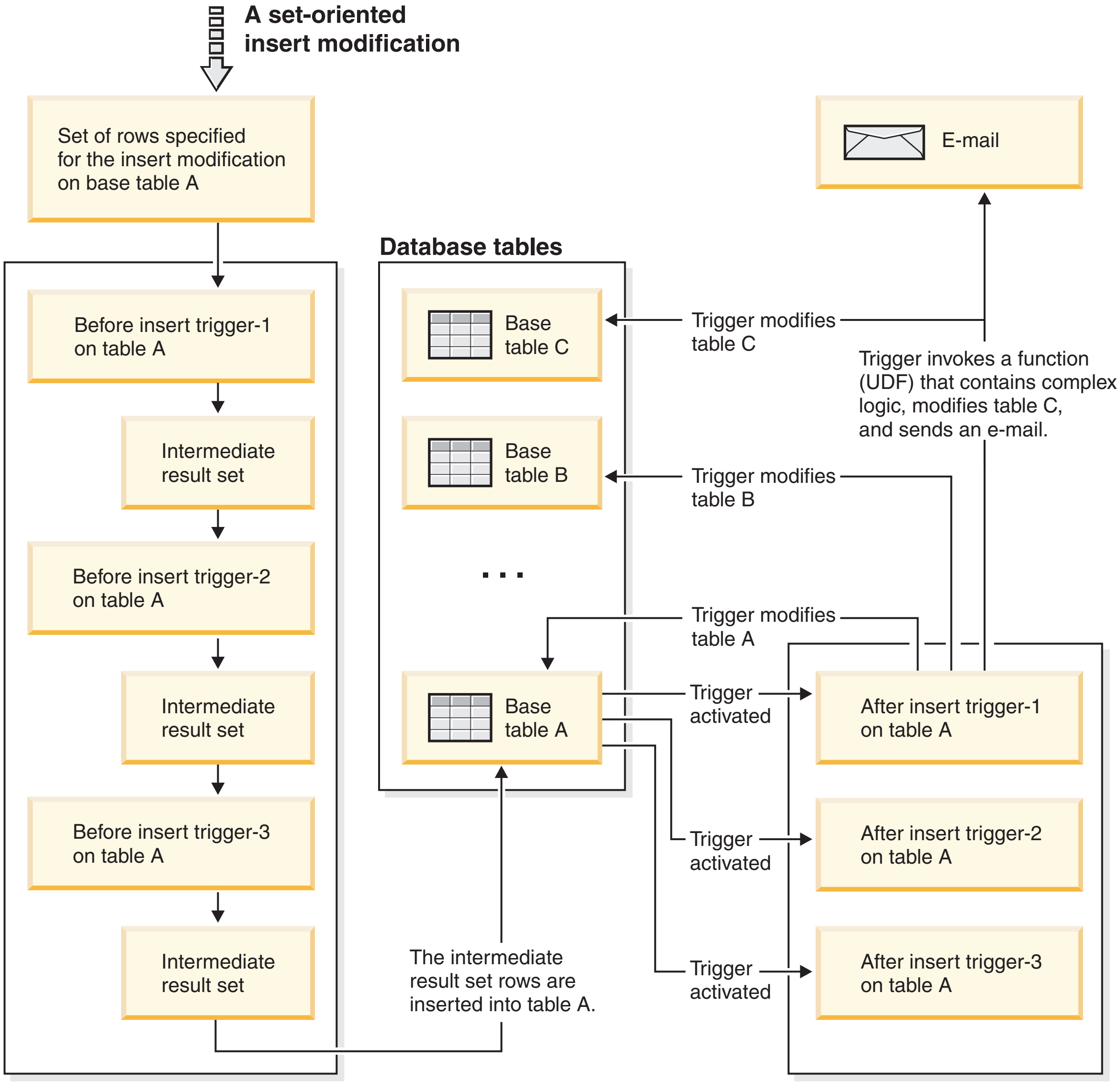

A set-oriented insert modification
Set of rows specified for the insert modification on base table A
Before insert trigger-1 on table A
Intermediate result set
Before insert trigger-2 on table A
Intermediate result set
Before insert trigger-3 on table A
Intermediate result set
Database tables
Base table C
Base table B
. . .
Base table A
The intermediate result set rows are inserted into table A.
E-mail
Trigger modifies table C
Trigger invokes a function (UDF) that contains complex logic, modifies table C, and sends an e-mail.
Trigger modifies table B
Trigger modifies table A
Trigger activated
After insert trigger-1 on table A
Trigger activated
After insert trigger-2 on table A
Trigger activated
After insert trigger-3 on table A

Figure 33. Trigger execution model

For a given table with both before and AFTER triggers, and a modifying event that is associated with these triggers, all the BEFORE triggers are activated first. The first activated BEFORE trigger for a given event operates on the set of rows targeted by the operation and makes any update modifications to the set that its logic prescribes. The output of this BEFORE trigger is accepted as input by the next before-trigger. When all of the BEFORE triggers that are activated by the event have been fired, the intermediate result set, the result of the BEFORE trigger modifications to the rows targeted by the trigger event operation, is applied to the table. Then each AFTER trigger associated with the event is fired. The AFTER triggers might modify the same table, another table, or perform an action external to the database.

The different activation times of triggers reflect different purposes of triggers. Basically, BEFORE triggers are an extension to the constraint subsystem of the database management system. Therefore, you generally use them to:

- Perform validation of input data
- Automatically generate values for newly inserted rows

- Read from other tables for cross-referencing purposes

BEFORE triggers are not used for further modifying the database because they are activated before the trigger event is applied to the database. Consequently, they are activated before integrity constraints are checked.

Conversely, you can view AFTER triggers as a module of application logic that runs in the database every time a specific event occurs. As a part of an application, AFTER triggers always see the database in a consistent state. Note that they are run after the integrity constraint validations. Consequently, you can use them mostly to perform operations that an application can also perform. For example:

- Perform follow on modify operations in the database.
- Perform actions outside the database, for example, to support alerts. Note that actions performed outside the database are not rolled back if the trigger is rolled back.

In contrast, you can view an INSTEAD OF trigger as a description of the inverse operation of the view it is defined on. For example, if the select list in the view contains an expression over a table, the INSERT statement in the body of its INSTEAD OF INSERT trigger will contain the reverse expression.

Because of the different nature of BEFORE, AFTER, and INSTEAD OF triggers, a different set of SQL operations can be used to define the triggered actions of BEFORE and AFTER, INSTEAD OF triggers. For example, update operations are not allowed in BEFORE triggers because there is no guarantee that integrity constraints will not be violated by the triggered action. Similarly, different trigger granularities are supported in BEFORE, AFTER, and INSTEAD OF triggers.

The triggered SQL statement of all triggers can be a dynamic compound statement. However, BEFORE triggers face some restrictions; they cannot contain the following SQL statements:

- UPDATE
- DELETE
- INSERT
- MERGE

Defining conditions for when trigger-action will fire (WHEN clause)

The activation of a trigger results in the running of its associated triggered action. Every trigger has exactly one triggered action which, in turn, has two components: an optional *triggered action condition* or WHEN clause, and a set of *triggered statement(s)*.

About this task

The *triggered action condition* is an optional clause of the triggered action which specifies a search condition that must evaluate to true to run statements within the triggered action. If the WHEN clause is omitted, then the statements within the triggered action are always executed.

The triggered action condition is evaluated once for each row if the trigger is a FOR EACH ROW trigger, and once for the statement if the trigger is a FOR EACH STATEMENT trigger.

This clause provides further control that you can use to fine tune the actions activated on behalf of a trigger. An example of the usefulness of the WHEN clause is to enforce a data dependent rule in which a triggered action is activated only if the incoming value falls inside or outside of a certain range.

The activation of a trigger results in the running of its associated triggered action. Every trigger has exactly one triggered action which, in turn, has two components:

The triggered action condition defines whether or not the set of triggered statements are performed for the row or for the statement for which the triggered action is executing. The set of triggered statements define the set of actions performed by the trigger in the database as a consequence of its event having occurred.

Example

For example, the following trigger action specifies that the set of triggered statements should only be activated for rows in which the value of the on_hand column is less than ten per cent of the value of the `max_stocked` column. In this case, the set of triggered statements is the invocation of the `issue_ship_request` function.

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
  FOR EACH ROW

  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
  BEGIN ATOMIC
    VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                              N_ROW.ON_HAND,
                              N_ROW.PARTNO));
  END
```

The set of triggered statements carries out the real actions caused by activating a trigger. Not every SQL operation is meaningful in every trigger. Depending on whether the trigger activation time is BEFORE or AFTER, different kinds of operations might be appropriate as a triggered statement.

In most cases, if any triggered statement returns a negative return code, the triggering statement together with all trigger and referential constraint actions are rolled back. The trigger name, SQLCODE, SQLSTATE and many of the tokens from the failing triggered statement are returned in the error message.

Supported SQL PL statements in triggers

The triggered SQL statement of all triggers can be a dynamic compound statement.

That is, triggered SQL statements can contain one or more of the following elements:

- CALL statement
- DECLARE variable statement
- SET variable statement
- WHILE loop
- FOR loop
- IF statement
- SIGNAL statement

- ITERATE statement
- LEAVE statement
- GET DIGNOSTIC statement
- fullselect

However, only AFTER and INSTEAD OF triggers can contain one or more of the following SQL statements:

- UPDATE statement
- DELETE statement
- INSERT statement
- MERGE statement

Accessing old and new column values in triggers using transition variables

When you implement a FOR EACH ROW trigger, it might be necessary to refer to the value of columns of the row in the set of affected rows, for which the trigger is currently executing. Note that to refer to columns in tables in the database (including the subject table), you can use regular SELECT statements.

About this task

A FOR EACH ROW trigger can refer to the columns of the row for which it is currently executing by using two transition variables that you can specify in the REFERENCING clause of a CREATE TRIGGER statement. There are two kinds of transition variables, which are specified as OLD and NEW, together with a correlation-name. They have the following semantics:

OLD AS correlation-name
Specifies a correlation name which captures the original state of the row, that is, before the triggered action is applied to the database.

NEW AS correlation-name
Specifies a correlation name which captures the value that is, or was, used to update the row in the database when the triggered action is applied to the database.

Example

Consider the following example:

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
  FOR EACH ROW
  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED
  AND N_ROW.ORDER_PENDING = 'N')
  BEGIN ATOMIC
    VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                              N_ROW.ON_HAND,
                              N_ROW.PARTNO));
    UPDATE PARTS SET PARTS.ORDER_PENDING = 'Y'
    WHERE PARTS.PARTNO = N_ROW.PARTNO;
  END
```

What to do next

Based on the definition of the OLD and NEW transition variables given previously, it is clear that not every transition variable can be defined for every trigger. Transition variables can be defined depending on the kind of trigger event:

UPDATE
An UPDATE trigger can refer to both OLD and NEW transition variables.

INSERT
An INSERT trigger can only refer to a NEW transition variable because before the activation of the INSERT operation, the affected row does not exist in the database. That is, there is no original state of the row that would define old values before the triggered action is applied to the database.

DELETE
A DELETE trigger can only refer to an OLD transition variable because there are no new values specified in the delete operation.

Note: Transition variables can only be specified for FOR EACH ROW triggers. In a FOR EACH STATEMENT trigger, a reference to a transition variable is not sufficient to specify to which of the several rows in the set of affected rows the transition variable is referring. Instead, refer to the set of new and old rows by using the OLD TABLE and NEW TABLE clauses of the CREATE TRIGGER statement. For more information about these clauses, see the CREATE TRIGGER statement.

Referencing old and new table result sets using transition tables

In both FOR EACH ROW and FOR EACH STATEMENT triggers, it might be necessary to refer to the whole set of affected rows. This is necessary, for example, if the trigger body needs to apply aggregations over the set of affected rows (for example, MAX, MIN, or AVG of some column values).

About this task

A trigger can refer to the set of affected rows by using two transition tables that can be specified in the REFERENCING clause of a CREATE TRIGGER statement. Just like the transition variables, there are two kinds of transition tables, which are specified as OLD_TABLE and NEW_TABLE together with a table-name, with the following semantics:

OLD_TABLE AS table-name
Specifies the name of the table which captures the original state of the set of affected rows (that is, before the triggering SQL operation is applied to the database).

NEW_TABLE AS table-name
Specifies the name of the table which captures the value that is used to update the rows in the database when the triggered action is applied to the database.

Example

For example:

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW_TABLE AS N_TABLE
  NEW AS N_ROW
  FOR EACH ROW
  WHEN ((SELECT AVG (ON_HAND) FROM N_TABLE) > 35)
  BEGIN ATOMIC
    VALUES(INFORM_SUPERVISOR(N_ROW.PARTNO,
                             N_ROW.MAX_STOCKED,
                             N_ROW.ON_HAND));
  END
```

Note that NEW_TABLE always has the full set of updated rows, even on a FOR EACH ROW trigger. When a trigger acts on the table on which the trigger is defined, NEW_TABLE contains the changed rows from the statement that activated the trigger. However, NEW_TABLE does not contain the changed rows that were caused by statements within the trigger, as that would cause a separate activation of the trigger.

What to do next

The transition tables are read-only. The same rules that define the kinds of transition variables that can be defined for which trigger event, apply for transition tables:

UPDATE
An UPDATE trigger can refer to both OLD_TABLE and NEW_TABLE transition tables.

INSERT
An INSERT trigger can only refer to a NEW_TABLE transition table because before the activation of the INSERT operation the affected rows do not exist in the database. That is, there is no original state of the rows that defines old values before the triggered action is applied to the database.

DELETE
A DELETE trigger can only refer to an OLD_TABLE transition table because there are no new values specified in the delete operation.

Note: It is important to observe that transition tables can be specified for both granularities of AFTER triggers: FOR EACH ROW and FOR EACH STATEMENT.

The scope of the `OLD_TABLE` and `NEW_TABLE` `table-name` is the trigger body. In this scope, this name takes precedence over the name of any other table with the same unqualified *table-name* that might exist in the schema. Therefore, if the `OLD_TABLE` or `NEW_TABLE` `table-name` is for example, X, a reference to X (that is, an unqualified X) in the FROM clause of a SELECT statement will always refer to the transition table even if there is a table named X in the in the schema of the trigger creator. In this case, the user has to make use of the fully qualified name in order to refer to the table X in the schema.

Creating triggers

A trigger defines a set of actions that are executed with, or triggered by, an INSERT, UPDATE, or DELETE clause on a specified table or a typed table.

About this task

Use triggers to:

- Validate input data
- Generate a value for a newly inserted row
- Read from other tables for cross-referencing purposes
- Write to other tables for audit-trail purposes

You can use triggers to support general forms of integrity or business rules. For example, a trigger can check a customer's credit limit before an order is accepted or update a summary data table.

Benefits:

- Faster application development: Because a trigger is stored in the database, you do not have to code the actions that it performs in every application.
- Easier maintenance: After a trigger is defined, it is automatically invoked when the table that it is created on is accessed.
- Global enforcement of business rules: If a business policy changes, you only need to change the trigger and not each application program.

When creating an atomic trigger, care must be taken with the end-of-statement character. The command line processor, by default, considers a “;” the end-of-statement marker. You should manually edit the end-of-statement character in your script to create the atomic trigger so that you are using a character other than “;”. For example, the “;” can be replaced by another special character like “#”. You can also precede the CREATE TRIGGER DDL with:

```
--#SET TERMINATOR @
```

To change the terminator in the CLP on the fly, the following syntax sets it back:

```
--#SET TERMINATOR
```

To create a trigger from the command line, enter:

```
db2 -td delimiter -vf script
```

where the *delimiter* is the alternative end-of-statement character and the *script* is the modified script with the new *delimiter* in it.

A trigger body can include one or more of the following statements: INSERT, searched UPDATE, searched DELETE, fullselect, SET Variable, and SIGNAL SQLSTATE. The trigger can be activated before or after the INSERT, UPDATE, or DELETE statement to which it refers.

Restrictions

- You cannot use triggers with nicknames.
- If the trigger is a BEFORE trigger, the column name specified by the triggered action must not be a generated column other than an identity column. That is, the generated identity value is visible to BEFORE triggers.

Procedure

To create a trigger from the command line, enter:

```
CREATE TRIGGER name
   action ON table_name
   operation
   triggered_action
```

Example

The following statement creates a trigger that increases the number of employees each time a new person is hired, by adding 1 to the number of employees (NBEMP) column in the COMPANY_STATS table each time a row is added to the EMPLOYEE table.

```
CREATE TRIGGER NEW_HIRED
   AFTER INSERT ON EMPLOYEE
   FOR EACH ROW
   UPDATE COMPANY_STATS SET NBEMP = NBEMP+1;
```

Modifying and dropping triggers

Triggers cannot be modified. They must be dropped and then created again according to the new definitions you require.

Before you begin

Trigger dependencies

- All dependencies of a trigger on some other object are recorded in the SYSCAT.TRIGDEP system catalog view. A trigger can depend on many objects.
- If an object that a trigger is dependent on is dropped, the trigger becomes inoperative but its definition is retained in the system catalog view. To re-validate this trigger, you must retrieve its definition from the system catalog view and submit a new CREATE TRIGGER statement.
- If a trigger is dropped, its description is deleted from the SYSCAT.TRIGGERS system catalog view and all of its dependencies are deleted from the SYSCAT.TRIGDEP system catalog view. All packages having UPDATE, INSERT, or DELETE dependencies on the trigger are invalidated.
- If the view is dependent on the trigger and it is made inoperative, the trigger is also marked inoperative. Any packages dependent on triggers that have been marked inoperative are invalidated.

About this task

A trigger object can be dropped using the DROP TRIGGER statement, but this procedure will cause dependent packages to be marked invalid, as follows:

- If an update trigger without an explicit column list is dropped, then packages with an update usage on the target table are invalidated.
- If an update trigger with a column list is dropped, then packages with update usage on the target table are only invalidated if the package also had an update usage on at least one column in the column-name list of the CREATE TRIGGER statement.
- If an insert trigger is dropped, packages that have an insert usage on the target table are invalidated.
- If a delete trigger is dropped, packages that have a delete usage on the target table are invalidated.

A package remains invalid until the application program is explicitly bound or rebound, or it is run and the database manager automatically rebinds it.

Examples of triggers and trigger use

Examples of interaction between triggers and referential constraints

Update operations can cause the interaction of triggers with referential constraints and check constraints.

Figure 24 on page 308 and the associated description are representative of the processing that is performed for an statement that updates data in the database.

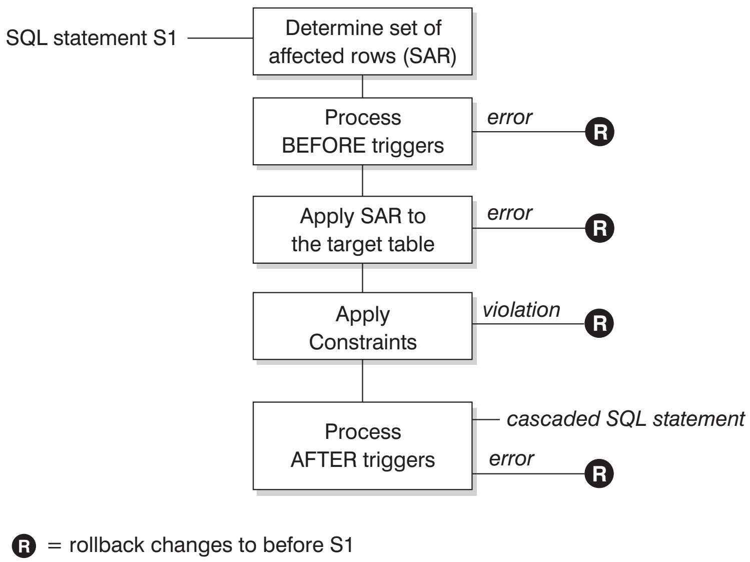


Figure 34. Processing an statement with associated triggers and constraints

Figure 24 on page 308 shows the general order of processing for an statement that updates a table. It assumes a situation where the table includes BEFORE triggers, referential constraints, check constraints and AFTER triggers that cascade. The following is a description of the boxes and other items found in Figure 24 on page 308.

- statement S_1

  This is the DELETE, INSERT, or UPDATE statement that begins the process. The statement S_1 identifies a table (or an updatable view over some table) referred to as the *subject table* throughout this description.

- Determine set of affected rows

  This step is the starting point for a process that repeats for referential constraint delete rules of CASCADE and SET NULL and for cascaded statements from AFTER triggers.

  The purpose of this step is to determine the *set of affected rows* for the statement. The set of rows included is based on the statement:

  - for DELETE, all rows that satisfy the search condition of the statement (or the current row for a positioned DELETE)
  - for INSERT, the rows identified by the VALUES clause or the fullselect

  - for UPDATE, all rows that satisfy the search condition (or the current row for a positioned UPDATE).

  If the set of affected rows is empty, there will be no BEFORE triggers, changes to apply to the subject table, or constraints to process for the statement.

- Process BEFORE triggers

  All BEFORE triggers are processed in ascending order of creation. Each BEFORE trigger will process the triggered action once for each row in the set of affected rows.

  An error can occur during the processing of a triggered action in which case all changes made as a result of the original statement S_1 (so far) are rolled back.

  If there are no BEFORE triggers or the set of affected is empty, this step is skipped.

- Apply the set of affected rows to the subject table

  The actual delete, insert, or update is applied using the set of affected rows to the subject table in the database.

  An error can occur when applying the set of affected rows (such as attempting to insert a row with a duplicate key where a unique index exists) in which case all changes made as a result of the original statement S_1 (so far) are rolled back.

- Apply Constraints

  The constraints associated with the subject table are applied if set of affected rows is not empty. This includes unique constraints, unique indexes, referential constraints, check constraints and checks related to the WITH CHECK OPTION on views. Referential constraints with delete rules of cascade or set null might cause additional triggers to be activated.

  A violation of any constraint or WITH CHECK OPTION results in an error and all changes made as a result of S_1 (so far) are rolled back.

- Process AFTER triggers

  All AFTER triggers activated by S_1 are processed in ascending order of creation.

  FOR EACH STATEMENT triggers will process the triggered action exactly once, even if the set of affected rows is empty. FOR EACH ROW triggers will process the triggered action once for each row in the set of affected rows.

  An error can occur during the processing of a triggered action in which case all changes made as a result of the original S_1 (so far) are rolled back.

  The triggered action of a trigger can include triggered statements that are DELETE, INSERT or UPDATE statements. For the purposes of this description, each such statement is considered a *cascaded statement*.

  A cascaded statement is a DELETE, INSERT, or UPDATE statement that is processed as part of the triggered action of an AFTER trigger. This statement starts a cascaded level of trigger processing. This can be thought of as assigning the triggered statement as a new S_1 and performing all of the steps described here recursively.

  Once all triggered statements from all AFTER triggers activated by each S_1 have been processed to completion, the processing of the original S_1 is completed.

- R = roll back changes to before S_1

  Any error (including constraint violations) that occurs during processing results in a roll back of all the changes made directly or indirectly as a result of the original statement S_1. The database is therefore back in the same state as immediately before the execution of the original statement S_1

Examples of defining actions using triggers

Assume that your general manager wants to keep the names of customers who have sent three or more complaints in the last 72 hours in a separate table. The general manager also wants to be informed whenever a customer name is inserted in this table more than once.

To define such actions, you define:

- An UNHAPPY_CUSTOMERS table:

```
CREATE TABLE UNHAPPY_CUSTOMERS (
  NAME           VARCHAR (30),
  EMAIL_ADDRESS  VARCHAR (200),
  INSERTION_DATE DATE)
```

- A trigger to automatically insert a row in UNHAPPY_CUSTOMERS if 3 or more messages were received in the last 3 days (assumes the existence of a CUSTOMERS table that includes a NAME column and an E_MAIL_ADDRESS column):

```
CREATE TRIGGER STORE_UNHAPPY_CUST
  AFTER INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (3 <= (SELECT COUNT(*)
              FROM ELECTRONIC_MAIL
              WHERE SENDER = N.SENDER
                AND SENDING_DATE(MESSAGE) > CURRENT DATE - 3 DAYS)
       )
  BEGIN ATOMIC
    INSERT INTO UNHAPPY_CUSTOMERS
    VALUES ((SELECT NAME
    FROM CUSTOMERS
    WHERE EMAIL_ADDRESS = N.SENDER), N.SENDER, CURRENT DATE);
  END
```

- A trigger to send a note to the general manager if the same customer is inserted in UNHAPPY_CUSTOMERS more than once (assumes the existence of a SEND_NOTE function that takes 2 character strings as input):

```
CREATE TRIGGER INFORM_GEN_MGR
  AFTER INSERT ON UNHAPPY_CUSTOMERS
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (1 <(SELECT COUNT(*)
            FROM UNHAPPY_CUSTOMERS
            WHERE EMAIL_ADDRESS = N.EMAIL_ADDRESS)
       )
  BEGIN ATOMIC
    VALUES(SEND_NOTE('Check customer:' CONCAT N.NAME,
                     'bigboss@vnet.ibm.com'));
  END
```

Example of defining business rules using triggers

Suppose your company has the policy that all email dealing with customer complaints must have Mr. Nelson, the marketing manager, in the carbon copy (CC) list.

Because this is a rule, you might want to express it as a constraint such as one of the following (assuming the existence of a CC_LIST UDF to check it):

```
ALTER TABLE ELECTRONIC_MAIL ADD
  CHECK (SUBJECT <> 'Customer complaint' OR
         CONTAINS (CC_LIST(MESSAGE), 'nelson@vnet.ibm.com') = 1)
```

However, such a constraint prevents the insertion of email dealing with customer complaints that do not have the marketing manager in the cc list. This is certainly not the intent of your company's business rule. The intent is to forward to the marketing manager any email dealing with customer complaints that were not copied to the marketing manager. Such a business rule can only be expressed with a trigger because it requires taking actions that cannot be expressed with declarative constraints. The trigger assumes the existence of a SEND_NOTE function with parameters of type E_MAIL and character string.

```
CREATE TRIGGER INFORM_MANAGER
  AFTER INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (N.SUBJECT = 'Customer complaint' AND
    CONTAINS (CC_LIST(MESSAGE), 'nelson@vnet.ibm.com') = 0)
  BEGIN ATOMIC
    VALUES(SEND_NOTE(N.MESSAGE, 'nelson@vnet.ibm.com'));
  END
```

Example of preventing operations on tables using triggers

Suppose you want to prevent undeliverable email from being stored in a table named ELECTRONIC_MAIL. To do so, you must prevent the execution of certain SQL INSERT statements.

There are two ways to do this:

- Define a BEFORE trigger that returns an error whenever the subject of an email is *undelivered mail*:

```
CREATE TRIGGER BLOCK_INSERT
  NO CASCADE BEFORE INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (SUBJECT(N.MESSAGE) = 'undelivered mail')
  BEGIN ATOMIC
    SIGNAL SQLSTATE '85101'
      SET MESSAGE_TEXT = ('Attempt to insert undelivered mail');
     END
```

- Define a check constraint forcing values of the new column SUBJECT to be different from *undelivered mail*:

```
ALTER TABLE ELECTRONIC_MAIL
  ADD CONSTRAINT NO_UNDELIVERED
  CHECK (SUBJECT <> 'undelivered mail')
```

Chapter 24. Sequences

A *sequence* is a database object that allows the automatic generation of values, such as cheque numbers. Sequences are ideally suited to the task of generating unique key values. Applications can use sequences to avoid possible concurrency and performance problems resulting from column values used to track numbers. The advantage that sequences have over numbers created outside the database is that the database server keeps track of the numbers generated. A crash and restart will not cause duplicate numbers from being generated.

The sequence numbers generated have the following properties:

- Values can be any exact numeric data type with a scale of zero. Such data types include: SMALLINT, BIGINT, INTEGER, and DECIMAL.
- Consecutive values can differ by any specified integer increment. The default increment value is 1.
- Counter value is recoverable. The counter value is reconstructed from logs when recovery is required.
- Values can be cached to improve performance. Pre-allocating and storing values in the cache reduces synchronous I/O to the log when values are generated for the sequence. In the event of a system failure, all cached values that have not been used are considered lost. The value specified for CACHE is the maximum number of sequence values that could be lost.

There are two expressions that can used with sequences:

- **NEXT VALUE expression:** returns the next value for the specified sequence. A new sequence number is generated when a NEXT VALUE expression specifies the name of the sequence. However, if there are multiple instances of a NEXT VALUE expression specifying the same sequence name within a query, the counter for the sequence is incremented only once for each row of the result, and all instances of NEXT VALUE return the same value for each row of the result.
- **PREVIOUS VALUE expression:** returns the most recently generated value for the specified sequence for a previous statement within the current application process. That is, for any given connection, the PREVIOUS VALUE remains constant even if another connection invokes NEXT VALUE.

For complete details and examples of these expressions, see “Sequence reference” in *SQL Reference Volume 1*.

Designing sequences

When designing sequences you must consider the differences between identity columns and sequences, and which is more appropriate for your environment. If you decide to use sequences, you must be familiar with the available options and parameters.

About this task

Before designing sequences, see “Sequences compared to identity columns” on page 370.

In addition to being simple to set up and create, the sequence has a variety of additional options that allows you more flexibility in generating the values:

- Choose from a variety of data types (SMALLINT, INTEGER, BIGINT, DECIMAL)
- Change starting values (START WITH)
- Change the sequence increment, including specifying increasing or decreasing values (INCREMENT BY)
- Set minimum and maximum values where the sequence numbering starts and stops (MINVALUE/MAXVALUE)
- Allow wrapping of values so that sequences can start over again, or disallow cycling (CYCLE/NO CYCLE)
- Allow caching of sequence values to improve performance, or disallow caching(CACHE/NO CACHE)

Even after the sequence has been generated, many of these values can be altered. For instance, you might want to set a different starting value depending on the day of the week. Another practical example of using sequences is for the generation and processing of bank checks. The sequence of bank check numbers is extremely important, and there are serious consequences if a batch of sequence numbers is lost or corrupted.

For improved performance, you should also be aware of and make use of the CACHE option. This option tells the database manager how many sequence values should be generated by the system before going back to the catalog to generate another set of sequences. The default CACHE value is 20, if not specified. Using the default as an example, the database manager automatically generates 20 sequential values in memory (1, 2, ..., 20) when the first sequence value is requested. Whenever a new sequence number is required, this memory cache of values is used to return the next value. Once this cache of values is used up, the database manager will generate the next twenty values (21, 22, ..., 40).

By implementing caching of sequence numbers, the database manager does not have to continually go to the catalog tables to get the next value. This reduces the extra processing associated with retrieving sequence numbers, but it also leads to possible gaps in the sequences if a system failure occurs, or if the system is shut down. For instance, if you decide to set the sequence cache to 100, the database manager will cache 100 values of these numbers and also set the system catalog to show that the next sequence of values should begin at 201. In the event that the database is shut down, the next set of sequence numbers will begin at 201. The numbers that were generated from 101 to 200 will be lost from the set of sequences if they were not used. If gaps in generated values cannot be tolerated in your application, you must set the caching value to NO CACHE despite the higher system overhead this will cause.

For more information about all available options and associated values, see the CREATE SEQUENCE statement.

Managing sequence behavior

You can tailor the behavior of sequences to meet the needs of your application. You change the attributes of a sequence when you issue the CREATE SEQUENCE statement to create a new sequence, and when you issue the ALTER SEQUENCE statement for an existing sequence.

Following are some of the attributes of a sequence that you can specify:

Data type

The AS clause of the CREATE SEQUENCE statement specifies the numeric data type of the sequence. The data type determines the possible minimum and maximum values of the sequence. The minimum and maximum values for a data type are listed in the *SQL Reference*. You cannot change the data type of a sequence; instead, you must drop the sequence by issuing the DROP SEQUENCE statement and issue a CREATE SEQUENCE statement with the new data type.

Start value

The START WITH clause of the CREATE SEQUENCE statement sets the initial value of the sequence. The RESTART WITH clause of the ALTER SEQUENCE statement resets the value of the sequence to a specified value.

Minimum value

The MINVALUE clause sets the minimum value of the sequence.

Maximum value

The MAXVALUE clause sets the maximum value of the sequence.

Increment value

The INCREMENT BY clause sets the value that each NEXT VALUE expression adds to the current value of the sequence. To decrement the value of the sequence, specify a negative value.

Sequence cycling

The CYCLE clause causes the value of a sequence that reaches its maximum or minimum value to generate its respective minimum value or maximum value on the following NEXT VALUE expression.

Note: CYCLE should only be used if unique numbers are not required or if it can be guaranteed that older sequence values are not in use anymore once the sequence cycles.

For example, to create a sequence called `id_values` that starts with a minimum value of 0, has a maximum value of 1000, increments by 2 with each NEXT VALUE expression, and returns to its minimum value when the maximum value is reached, issue the following statement:

```
CREATE SEQUENCE id_values
  START WITH 0
  INCREMENT BY 2
  MAXVALUE 1000
  CYCLE
```

Application performance and sequences

Like identity columns, using sequences to generate values generally improves the performance of your applications in comparison to alternative approaches. The alternative to sequences is to create a single-column table that stores the current value and to increment that value with either a trigger or under the control of the application. However, in a distributed environment where applications concurrently access the single-column table, the locking required to force serialized access to the table can seriously affect performance.

Sequences avoid the locking issues that are associated with the single-column table approach and can cache sequence values in memory to improve response time. To maximize the performance of applications that use sequences, ensure that your sequence caches an appropriate amount of sequence values. The CACHE clause of the CREATE SEQUENCE and ALTER SEQUENCE statements specifies the maximum number of sequence values that the database manager generates and stores in memory.

If your sequence must generate values in order, without introducing gaps in that order because of a system failure or database deactivation, use the ORDER and NO CACHE clauses in the CREATE SEQUENCE statement. The NO CACHE clause guarantees that no gaps appear in the generated values at the cost of some of your application's performance because it forces your sequence to write to the database log every time it generates a new value. Note that gaps can still appear due to transactions that rollback and do not actually use that sequence value that they requested.

Sequences compared to identity columns

Although sequences and identity columns seem to serve similar purposes for DB2 applications, there is an important difference. An identity column automatically generates values for a column in a single table using the **LOAD** utility. A sequence generates sequential values upon request that can be used in any SQL statement using the CREATE SEQUENCE statement.

Identity columns

Allow the database manager to automatically generate a unique numeric value for each row that is added to the table. If you are creating a table and you know you need to uniquely identify each row that is added to that table, then you can add an identity column to the table definition as part of the CREATE TABLE statement:

```
CREATE TABLE table_name
  (column_name_1 INT,
   column_name_2, DOUBLE,
   column_name_3 INT NOT NULL GENERATED ALWAYS AS IDENTITY
     (START WITH value_1, INCREMENT BY value_2))
```

In this example, the third column identifies the identity column. One of the attributes that you can define is the value used in the column to uniquely define each row when a row is added. The value following the INCREMENT BY clause shows by how much subsequent values of the identity column contents increase for every row added to the table.

After they are created, the identity properties can be changed or removed using the ALTER TABLE statement. You can also use the ALTER TABLE statement to add identity properties on other columns.

Sequences

Allow the automatic generation of values. Sequences are ideally suited to the task of generating unique key values. Applications can use sequences to avoid possible concurrency and performance problems resulting from the generation of a unique counter through other means. Unlike an identity column, a sequence is not tied to a particular table column, nor is it bound to a unique table column and only accessible through that table column.

A sequence can be created, and later altered, so that it generates values by incrementing or decrementing values either without a limit; or to a user-defined limit, and then stopping; or to a user-defined limit, then cycling back to the beginning and starting again.

The following example shows how to create a sequence called `orderseq`:

```
CREATE SEQUENCE orderseq
  START WITH 1
  INCREMENT BY 1
  NOMAXVALUE
  NOCYCLE
  CACHE 50
```

In this example, the sequence starts at 1 and increases by 1 with no upper limit. There is no reason to cycle back to the beginning and restart from 1 because there is no assigned upper limit. The CACHE parameter specifies the maximum number of sequence values that the database manager preallocates and keeps in memory.

Creating sequences

To create sequences, use the CREATE SEQUENCE statement. Unlike an identity column attribute, a sequence is not tied to a particular table column nor is it bound to a unique table column and only accessible through that table column.

About this task

There are several restrictions on where NEXT VALUE or PREVIOUS VALUE expressions can be used. A sequence can be created, or altered, so that it generates values in one of these ways:

- Increment or decrement monotonically (changing by a constant amount) without bound
- Increment or decrement monotonically to a user-defined limit and stop
- Increment or decrement monotonically to a user-defined limit and cycle back to the beginning and start again

Note: Use caution when recovering databases that use sequences: For sequence values that are used outside the database, for example sequence numbers used for bank checkes, if the database is recovered to a point in time before the database failure, then this could cause the generation of duplicate values for some sequences. To avoid possible duplicate values, databases that use sequence values outside the database should not be recovered to a prior point in time.

To create a sequence called `order_seq` using defaults for all the options, issue the following statement in an application program or through the use of dynamic SQL statements:

```
CREATE SEQUENCE order_seq
```

This sequence starts at 1 and increases by 1 with no upper limit.

This example could represent processing for a batch of bank checks starting from 101 to 200. The first order would have been from 1 to 100. The sequence starts at 101 and increase by 1 with an upper limit of 200. NOCYCLE is specified so that duplicate cheque numbers are not produced. The number associated with the CACHE parameter specifies the maximum number of sequence values that the database manager preallocates and keeps in memory.

```
CREATE SEQUENCE order_seq
   START WITH 101
   INCREMENT BY 1
   MAXVALUE 200
   NOCYCLE
   CACHE 25
```

For more information about these and other options, and authorization requirements, see the CREATE SEQUENCE statement.

Generating sequential values

Generating sequential values is a common database application development problem. The best solution to that problem is to use sequences and sequence expressions in SQL. Each *sequence* is a uniquely named database object that can be accessed only by sequence expressions.

There are two *sequence expressions*: the PREVIOUS VALUE expression and the NEXT VALUE expression. The PREVIOUS VALUE expression returns the value most recently generated in the application process for the specified sequence. Any NEXT VALUE expressions occurring in the same statement as the PREVIOUS VALUE expression have no effect on the value generated by the PREVIOUS VALUE expression in that statement. The NEXT VALUE sequence expression increments the value of the sequence and returns the new value of the sequence.

To create a sequence, issue the CREATE SEQUENCE statement. For example, to create a sequence called `id_values` using the default attributes, issue the following statement:

```
  CREATE SEQUENCE id_values
```

To generate the first value in the application session for the sequence, issue a VALUES statement using the NEXT VALUE expression:

```
VALUES NEXT VALUE FOR id_values

1
-----------
          1

            1 record(s) selected.
```

To update the value of a column with the next value of the sequence, include the NEXT VALUE expression in the UPDATE statement, as follows:

```
UPDATE staff
  SET id = NEXT VALUE FOR id_values
  WHERE id = 350
```

To insert a new row into a table using the next value of the sequence, include the NEXT VALUE expression in the INSERT statement, as follows:

```
INSERT INTO staff (id, name, dept, job)
  VALUES (NEXT VALUE FOR id_values, 'Kandil', 51, 'Mgr')
```

Determining when to use identity columns or sequences

Although there are similarities between identity columns and sequences, there are also differences. The characteristics of each can be used when designing your database and applications.

Depending on your database design and the applications using the database, the following characteristics will assist you in determining when to use identity columns and when to use sequences.

Identity column characteristics

- An identity column automatically generates values for a single table.

- When an identity column is defined as GENERATED ALWAYS, the values used are always generated by the database manager. Applications are not allowed to provide their own values during the modification of the contents of the table.
- After inserting a row, the generated identity value can be retrieved either by using the IDENTITY_VAL_LOCAL() function or by selecting the identity column back from the insert by using the SELECT FROM INSERT statement.
- The LOAD utility can generate IDENTITY values.

Sequence characteristics

- Sequences are not tied to any one table.
- Sequences generate sequential values that can be used in any SQL or XQuery statement.

  Since sequences can be used by any application, there are two expressions used to control the retrieval of the next value in the specified sequence and the value generated previous to the statement being executed. The PREVIOUS VALUE expression returns the most recently generated value for the specified sequence for a previous statement within the current session. The NEXT VALUE expression returns the next value for the specified sequence. The use of these expressions allows the same value to be used across several SQL and XQuery statements within several tables.

Sequence Modification

Modify the attributes of an existing sequence with the ALTER SEQUENCE statement.

The attributes of the sequence that can be modified include:

- Changing the increment between future values
- Establishing new minimum or maximum values
- Changing the number of cached sequence numbers
- Changing whether the sequence cycles or not
- Changing whether sequence numbers must be generated in order of request
- Restarting the sequence

There are two tasks that are not found as part of the creation of the sequence. They are:

- **RESTART:** Resets the sequence to the value specified implicitly or explicitly as the starting value when the sequence was created.
- **RESTART WITH *numeric-constant*:** Resets the sequence to the exact numeric constant value. The numeric constant can be any positive or negative value with no non-zero digits to the right of any decimal point.

Restrictions

The data type of a sequence cannot be changed. Instead, you must drop the current sequence and then create a sequence specifying the new data type.

After restarting a sequence or changing to CYCLE, it is possible to generate duplicate sequence numbers. Only future sequence numbers are affected by the ALTER SEQUENCE statement.

All cached sequence values not used by the database manager are lost when a sequence is altered.

Viewing sequence definitions

Use the VALUES statement using the PREVIOUS VALUE option to view the reference information associated with a sequence or to view the sequence itself.

Procedure

To display the current value of the sequence, issue a VALUES statement using the PREVIOUS VALUE expression:

```
VALUES PREVIOUS VALUE FOR id_values

1
-----------
          1

            1 record(s) selected.
```

You can repeatedly retrieve the current value of the sequence, and the value that the sequence returns does not change until you issue a NEXT VALUE expression. This is even true if another connection consumes sequence values at the same time.

Example

In the following example, the PREVIOUS VALUE expression returns a value of 1, until the NEXT VALUE expression in the current connection increments the value of the sequence:

```
VALUES PREVIOUS VALUE FOR id_values

1
-----------
          1

            1 record(s) selected.
VALUES PREVIOUS VALUE FOR id_values

1
-----------
          1

            1 record(s) selected.
VALUES NEXT VALUE FOR id_values

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

            1 record(s) selected.
VALUES PREVIOUS VALUE FOR id_values

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

            1 record(s) selected.
```

Dropping sequences

To delete a sequence, use the DROP statement.

Before you begin

When dropping sequences, the authorization ID of the statement must have DBADM authority.

Restrictions

Sequences that are system-created for IDENTITY columns cannot be dropped by using the DROP SEQUENCE statement.

Procedure

To drop a specific sequence, enter:

```
DROP SEQUENCE sequence_name
```

where the *sequence_name* is the name of the sequence to be dropped and includes the implicit or explicit schema name to exactly identify an existing sequence.

Results

Once a sequence is dropped, all privileges on the sequence are also dropped.

Examples of how to code sequences

Many applications that are written require the use of sequence number to track invoice numbers, customer numbers, and other objects which get incremented by one whenever a new item is required. The database manager can auto-increment values in a table through the use of identity columns. Although this technique works well for individual tables, it might not be the most convenient way of generating unique values that must be used across multiple tables.

The *sequence* object lets you create a value that gets incremented under programmer control and can be used across many tables. The following example shows a sequence number being created for customer numbers using a data type of integer:

```
CREATE SEQUENCE customer_no AS INTEGER
```

By default the sequence number starts at one and increments by one at a time and is of an INTEGER data type. The application needs to get the next value in the sequence by using the NEXT VALUE function. This function generates the next value for the sequence which can then be used for subsequent SQL statements:

```
VALUES NEXT VALUE FOR customer_no
```

Instead of generating the next number with the VALUES function, the programmer could have used this function within an INSERT statement. For instance, if the first column of the customer table contained the customer number, an INSERT statement could be written as follows:

```
INSERT INTO customers VALUES
  (NEXT VALUE FOR customer_no, 'comment', ...)
```

If the sequence number needs to be used for inserts into other tables, the PREVIOUS VALUE function can be used to retrieve the previously generated value. For instance, if the customer number just created needs to be used for a subsequent invoice record, the SQL would include the PREVIOUS VALUE function:

```
INSERT INTO invoices
  (34,PREVIOUS VALUE FOR customer_no, 234.44, ...)
```

The PREVIOUS VALUE function can be used multiple times within the application and it will only return the last value generated by that application. It might be possible that subsequent transactions have already incremented the sequence to another value, but you will always see the last number that is generated.

Sequence reference

A sequence reference is an expression which references a sequence defined at the application server.

sequence-reference:

```
├──┬┤ nextval-expression ├┬──────────────────────────────────────────┤
   └┤ prevval-expression ├┘
```

nextval-expression:

```
├──NEXT VALUE FOR──sequence-name──────────────────────────────────────┤
```

prevval-expression:

```
├──PREVIOUS VALUE FOR──sequence-name──────────────────────────────────┤
```

NEXT VALUE FOR *sequence-name*
A NEXT VALUE expression generates and returns the next value for the sequence specified by *sequence-name*.

PREVIOUS VALUE FOR *sequence-name*
A PREVIOUS VALUE expression returns the most recently generated value for the specified sequence for a previous statement within the current application process. This value can be referenced repeatedly by using PREVIOUS VALUE expressions that specify the name of the sequence. There may be multiple instances of PREVIOUS VALUE expressions specifying the same sequence name within a single statement; they all return the same value. In a partitioned database environment, a PREVIOUS VALUE expression may not return the most recently generated value.

A PREVIOUS VALUE expression can only be used if a NEXT VALUE expression specifying the same sequence name has already been referenced in the current application process, in either the current or a previous transaction (SQLSTATE 51035).

Notes

- ***Authorization:*** If a sequence-reference is used in a statement, the privileges held by the authorization ID of the statement must include at least one of the following privileges:

- The USAGE privilege on the sequence
- DATAACCESS authority

- A new value is generated for a sequence when a NEXT VALUE expression specifies the name of that sequence. However, if there are multiple instances of a NEXT VALUE expression specifying the same sequence name within a query, the counter for the sequence is incremented only once for each row of the result, and all instances of NEXT VALUE return the same value for a row of the result.
- The same sequence number can be used as a unique key value in two separate tables by referencing the sequence number with a NEXT VALUE expression for the first row (this generates the sequence value), and a PREVIOUS VALUE expression for the other rows (the instance of PREVIOUS VALUE refers to the sequence value most recently generated in the current session), as shown in the following example:

```
INSERT INTO order(orderno, cutno)
  VALUES (NEXT VALUE FOR order_seq, 123456);

INSERT INTO line_item (orderno, partno, quantity)
  VALUES (PREVIOUS VALUE FOR order_seq, 987654, 1);
```

- NEXT VALUE and PREVIOUS VALUE expressions can be specified in the following places:
  - select-statement or SELECT INTO statement (within the select-clause, provided that the statement does not contain a DISTINCT keyword, a GROUP BY clause, an ORDER BY clause, a UNION keyword, an INTERSECT keyword, or EXCEPT keyword)
  - INSERT statement (within a VALUES clause)
  - INSERT statement (within the select-clause of the fullselect)
  - UPDATE statement (within the SET clause (either a searched or a positioned UPDATE statement), except that NEXT VALUE cannot be specified in the select-clause of the fullselect of an expression in the SET clause)
  - SET Variable statement (except within the select-clause of the fullselect of an expression; a NEXT VALUE expression can be specified in a trigger, but a PREVIOUS VALUE expression cannot)
  - VALUES INTO statement (within the select-clause of the fullselect of an expression)
  - CREATE PROCEDURE statement (within the routine-body of an SQL procedure)
  - CREATE TRIGGER statement within the triggered-action (a NEXT VALUE expression may be specified, but a PREVIOUS VALUE expression cannot)
- NEXT VALUE and PREVIOUS VALUE expressions cannot be specified (SQLSTATE 428F9) in the following places:
  - Join condition of a full outer join
  - DEFAULT value for a column in a CREATE or ALTER TABLE statement
  - Generated column definition in a CREATE OR ALTER TABLE statement
  - Summary table definition in a CREATE TABLE or ALTER TABLE statement
  - Condition of a CHECK constraint
  - CREATE TRIGGER statement (a NEXT VALUE expression may be specified, but a PREVIOUS VALUE expression cannot)
  - CREATE VIEW statement
  - CREATE METHOD statement
  - CREATE FUNCTION statement
  - An argument list of an XMLQUERY, XMLEXISTS, or XMLTABLE expression

- In addition, a NEXT VALUE expression cannot be specified (SQLSTATE 428F9) in the following places:
  - CASE expression
  - Parameter list of an aggregate function
  - Subquery in a context other than those explicitly allowed, as described previously
  - SELECT statement for which the outer SELECT contains a DISTINCT operator
  - Join condition of a join
  - SELECT statement for which the outer SELECT contains a GROUP BY clause
  - SELECT statement for which the outer SELECT is combined with another SELECT statement using the UNION, INTERSECT, or EXCEPT set operator
  - Nested table expression
  - Parameter list of a table function
  - WHERE clause of the outer-most SELECT statement, or a DELETE or UPDATE statement
  - ORDER BY clause of the outer-most SELECT statement
  - select-clause of the fullselect of an expression, in the SET clause of an UPDATE statement
  - IF, WHILE, DO ... UNTIL, or CASE statement in an SQL routine
- When a value is generated for a sequence, that value is consumed, and the next time that a value is requested, a new value will be generated. This is true even when the statement containing the NEXT VALUE expression fails or is rolled back.

  If an INSERT statement includes a NEXT VALUE expression in the VALUES list for the column, and if an error occurs at some point during the execution of the INSERT (it could be a problem in generating the next sequence value, or a problem with the value for another column), then an insertion failure occurs (SQLSTATE 23505), and the value generated for the sequence is considered to be consumed. In some cases, reissuing the same INSERT statement might lead to success.

  For example, consider an error that is the result of the existence of a unique index for the column for which NEXT VALUE was used and the sequence value generated already exists in the index. It is possible that the next value generated for the sequence is a value that does not exist in the index and so the subsequent INSERT would succeed.
- ***Scope of PREVIOUS VALUE:*** The value of PREVIOUS VALUE persists until the next value is generated for the sequence in the current session, the sequence is dropped or altered, or the application session ends. The value is unaffected by COMMIT or ROLLBACK statements. The value of PREVIOUS VALUE cannot be directly set and is a result of executing the NEXT VALUE expression for the sequence.

  A technique commonly used, especially for performance, is for an application or product to manage a set of connections and route transactions to an arbitrary connection. In these situations, the availability of the PREVIOUS VALUE for a sequence should be relied on only until the end of the transaction. Examples of where this type of situation can occur include applications that use XA protocols, use connection pooling, use the connection concentrator, and use HADR to achieve failover.
- If in generating a value for a sequence, the maximum value for the sequence is exceeded (or the minimum value for a descending sequence) and cycles are not

permitted, then an error occurs (SQLSTATE 23522). In this case, the user could ALTER the sequence to extend the range of acceptable values, or enable cycles for the sequence, or DROP and CREATE a new sequence with a different data type that has a larger range of values.

For example, a sequence may have been defined with a data type of SMALLINT, and eventually the sequence runs out of assignable values. DROP and re-create the sequence with the new definition to redefine the sequence as INTEGER.

- A reference to a NEXT VALUE expression in the select statement of a cursor refers to a value that is generated for a row of the result table. A sequence value is generated for a NEXT VALUE expression for each row that is fetched from the database. If blocking is done at the client, the values may have been generated at the server before the processing of the FETCH statement. This can occur when there is blocking of the rows of the result table. If the client application does not explicitly FETCH all the rows that the database has materialized, then the application will not see the results of all the generated sequence values (for the materialized rows that were not returned).
- A reference to a PREVIOUS VALUE expression in the select statement of a cursor refers to a value that was generated for the specified sequence before the opening of the cursor. However, closing the cursor can affect the values returned by PREVIOUS VALUE for the specified sequence in subsequent statements, or even for the same statement in the event that the cursor is reopened. This would be the case when the select statement of the cursor included a reference to NEXT VALUE for the same sequence name.
- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - NEXTVAL and PREVVAL can be specified in place of NEXT VALUE and PREVIOUS VALUE
  - *sequence-name*.NEXTVAL can be specified in place of NEXT VALUE FOR *sequence-name*
  - *sequence-name*.CURRVAL can be specified in place of PREVIOUS VALUE FOR *sequence-name*

Examples

Assume that there is a table called "order", and that a sequence called "order_seq" is created as follows:

```
CREATE SEQUENCE order_seq
  START WITH 1
  INCREMENT BY 1
  NO MAXVALUE
  NO CYCLE
  CACHE 24
```

Following are some examples of how to generate an "order_seq" sequence number with a NEXT VALUE expression:

```
INSERT INTO order(orderno, custno)
  VALUES (NEXT VALUE FOR order_seq, 123456);
```

or

```
UPDATE order
  SET orderno = NEXT VALUE FOR order_seq
  WHERE custno = 123456;
```

or

```
VALUES NEXT VALUE FOR order_seq INTO :hv_seq;
```

Chapter 25. Views

A *view* is an efficient way of representing data without the need to maintain it. A view is not an actual table and requires no permanent storage. A "virtual table" is created and used.

A *view* provides a different way of looking at the data in one or more tables; it is a named specification of a result table. The specification is a SELECT statement that is run whenever the view is referenced in an SQL statement. A view has columns and rows just like a table. All views can be used just like tables for data retrieval. Whether a view can be used in an insert, update, or delete operation depends on its definition.

A view can include all or some of the columns or rows contained in the tables on which it is based. For example, you can join a department table and an employee table in a view, so that you can list all employees in a particular department.

Figure 35 shows the relationship between tables and views.

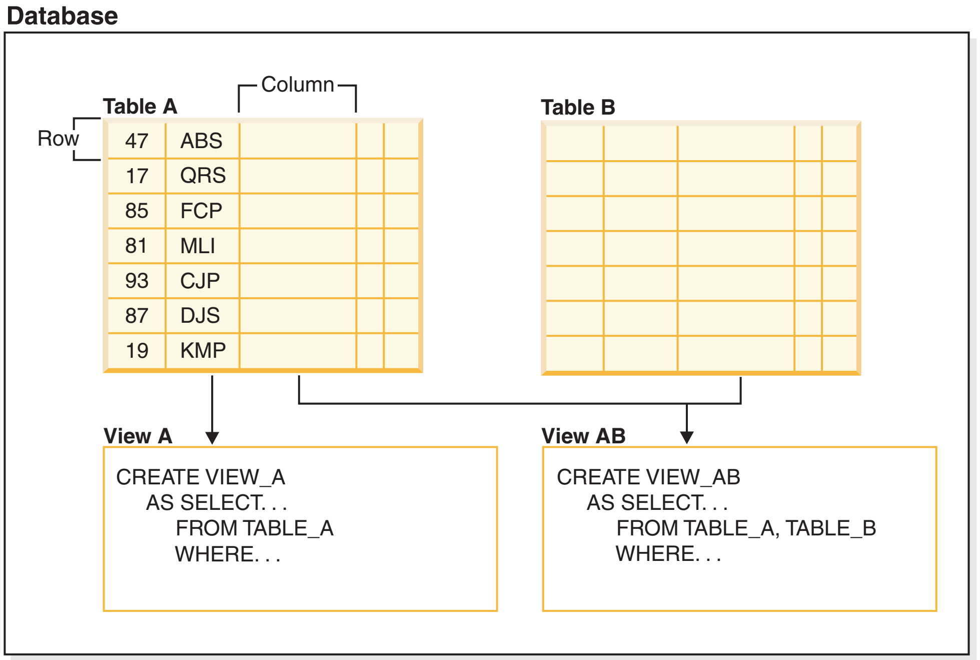


Figure 35. Relationship between tables and views

You can use views to control access to sensitive data, because views allow multiple users to see different presentations of the same data. For example, several users might be accessing a table of data about employees. A manager sees data about his or her employees but not employees in another department. A recruitment officer sees the hire dates of all employees, but not their salaries; a financial officer sees the salaries, but not the hire dates. Each of these users works with a view derived from the table. Each view appears to be a table and has its own name.

When the column of a view is directly derived from the column of a base table, that view column inherits any constraints that apply to the table column. For example, if a view includes a foreign key of its table, insert and update operations using that view are subject to the same referential constraints as is the table. Also, if the table of a view is a parent table, delete and update operations using that view are subject to the same rules as are delete and update operations on the table.

A view can derive the data type of each column from the result table, or base the types on the attributes of a user-defined structured type. This is called a *typed view*. Similar to a typed table, a typed view can be part of a view hierarchy. A *subview* inherits columns from its *superview*. The term *subview* applies to a typed view and to all typed views that are below it in the view hierarchy. A *proper subview* of a view V is a view below V in the typed view hierarchy.

A view can become inoperative (for example, if the table is dropped); if this occurs, the view is no longer available for SQL operations.

Designing views

A *view* provides a different way of looking at the data in one or more tables; it is a named specification of a result table.

The specification is a SELECT statement that is run whenever the view is referenced in an SQL statement. A view has columns and rows just like a base table. All views can be used just like tables for data retrieval. Whether a view can be used in an insert, update, or delete operation depends on its definition.

Views are classified by the operations they allow. They can be:

- Deletable
- Updatable
- Insertable
- Read-only

The view type is established according to its update capabilities. The classification indicates the kind of SQL operation that is allowed against the view.

Referential and check constraints are treated independently. They do not affect the view classification.

For example, you might not be able to insert a row into a table due to a referential constraint. If you create a view using that table, you also cannot insert that row using the view. However, if the view satisfies all the rules for an insertable view, it will still be considered an insertable view. This is because the insert restriction is on the table, not on the view definition.

For more information, see the CREATE VIEW statement.

System catalog views

The database manager maintains a set of tables and views that contain information about the data under its control. These tables and views are collectively known as the *system catalog*.

The system catalog contains information about the logical and physical structure of database objects such as tables, views, indexes, packages, and functions. It also contains statistical information. The database manager ensures that the descriptions in the system catalog are always accurate.

The system catalog views are like any other database view. SQL statements can be used to query the data in the system catalog views. A set of updatable system catalog views can be used to modify certain values in the system catalog.

Views with the check option

A view that is defined WITH CHECK OPTION enforces any rows that are modified or inserted against the SELECT statement for that view. Views with the check option are also called *symmetric views*. For example, a symmetric view that only returns only employees in department 10 will not allow insertion of employees in other departments. This option, therefore, ensures the integrity of the data being modified in the database, returning an error if the condition is violated during an INSERT or UPDATE operation.

If your application cannot define the required rules as table check constraints, or the rules do not apply to all uses of the data, there is another alternative to placing the rules in the application logic. You can consider creating a view of the table with the conditions on the data as part of the WHERE clause and the WITH CHECK OPTION clause specified. This view definition restricts the retrieval of data to the set that is valid for your application. Additionally, if you can update the view, the WITH CHECK OPTION clause restricts updates, inserts, and deletes to the rows applicable to your application.

The WITH CHECK OPTION must not be specified for the following views:

- Views defined with the read-only option (a read-only view)
- View that reference the NODENUMBER or PARTITION function, a nondeterministic function (for example, RAND), or a function with external action
- Typed views

Example 1

Following is an example of a view definition using the WITH CHECK OPTION. This option is required to ensure that the condition is always checked. The view ensures that the DEPT is always 10. This will restrict the input values for the DEPT column. When a view is used to insert a new value, the WITH CHECK OPTION is always enforced:

```
CREATE VIEW EMP_VIEW2
  (EMPNO, EMPNAME, DEPTNO, JOBTITLE, HIREDATE)
AS SELECT ID, NAME, DEPT, JOB, HIREDATE FROM EMPLOYEE
  WHERE DEPT=10
WITH CHECK OPTION;
```

If this view is used in an INSERT statement, the row will be rejected if the DEPTNO column is not the value 10. It is important to remember that there is no data validation during modification if the WITH CHECK OPTION is not specified.

If this view is used in a SELECT statement, the conditional (WHERE clause) would be invoked and the resulting table would only contain the matching rows of data. In other words, the WITH CHECK OPTION does not affect the result of a SELECT statement.

Example 2

With a view, you can make a subset of table data available to an application program and validate data that is to be inserted or updated. A view can have column names that are different from the names of corresponding columns in the original tables. For example:

```
CREATE VIEW <name> (<column>, <column>, <column>)
   SELECT <column_name> FROM <table_name>
   WITH CHECK OPTION
```

Example 3

The use of views provides flexibility in the way your programs and end-user queries can look at the table data.

The following SQL statement creates a view on the EMPLOYEE table that lists all employees in Department A00 with their employee and telephone numbers:

```
 CREATE VIEW EMP_VIEW (DA00NAME, DA00NUM, PHONENO)
    AS SELECT LASTNAME, EMPNO, PHONENO FROM EMPLOYEE
    WHERE WORKDEPT = 'A00'
    WITH CHECK OPTION
```

The first line of this statement names the view and defines its columns. The name EMP_VIEW must be unique within its schema in SYSCAT.TABLES. The view name appears as a table name although it contains no data. The view will have three columns called DA00NAME, DA00NUM, and PHONENO, which correspond to the columns LASTNAME, EMPNO, and PHONENO from the EMPLOYEE table. The column names listed apply one-to-one to the select list of the SELECT statement. If column names are not specified, the view uses the same names as the columns of the result table of the SELECT statement.

The second line is a SELECT statement that describes which values are to be selected from the database. It might include the clauses ALL, DISTINCT, FROM, WHERE, GROUP BY, and HAVING. The name or names of the data objects from which to select columns for the view must follow the FROM clause.

Example 4

The WITH CHECK OPTION clause indicates that any updated or inserted row to the view must be checked against the view definition, and rejected if it does not conform. This enhances data integrity but requires additional processing. If this clause is omitted, inserts and updates are not checked against the view definition.

The following SQL statement creates the same view on the EMPLOYEE table using the SELECT AS clause:

```
CREATE VIEW EMP_VIEW
   SELECT LASTNAME AS DA00NAME,
          EMPNO AS DA00NUM,
          PHONENO
   FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

For this example, the EMPLOYEE table might have salary information in it, which should not be made available to everyone. The employee's phone number, however, should be generally accessible. In this case, a view could be created from the LASTNAME and PHONENO columns only. Access to the view could be granted to PUBLIC, while access to the entire EMPLOYEE table could be restricted to those who have the authorization to see salary information.

Nested view definitions

If a view is based on another view, the number of predicates that must be evaluated is based on the WITH CHECK OPTION specification.

If a view is defined without WITH CHECK OPTION, the definition of the view is not used in the data validity checking of any insert or update operations. However, if the view directly or indirectly depends on another view defined with the WITH CHECK OPTION, the definition of that super view is used in the checking of any insert or update operation.

If a view is defined with the WITH CASCADED CHECK OPTION or just the WITH CHECK OPTION (CASCADED is the default value of the WITH CHECK OPTION), the definition of the view is used in the checking of any insert or update operations. In addition, the view inherits the search conditions from any updatable views on which the view depends. These conditions are inherited even if those views do not include the WITH CHECK OPTION. Then the inherited conditions are multiplied together to conform to a constraint that is applied for any insert or update operations for the view or any views depending on the view.

As an example, if a view V2 is based on a view V1, and the check option for V2 is defined with the WITH CASCADED CHECK OPTION, the predicates for both views are evaluated when INSERT and UPDATE statements are performed against the view V2:

```
CREATE VIEW EMP_VIEW2 AS
  SELECT EMPNO, EMPNAME, DEPTNO FROM EMP
    WHERE DEPTNO = 10
WITH CHECK OPTION;
```

The following example shows a CREATE VIEW statement using the WITH CASCADED CHECK OPTION. The view EMP_VIEW3 is created based on a view EMP_VIEW2, which has been created with the WITH CHECK OPTION. If you want to insert or update a record to EMP_VIEW3, the record should have the values DEPTNO=10 and EMPNO=20.

```
CREATE VIEW EMP_VIEW3 AS
  SELECT EMPNO, EMPNAME, DEPTNO FROM EMP_VIEW2
    WHERE EMPNO > 20
WITH CASCADED CHECK OPTION;
```

Note: The condition DEPTNO=10 is enforced for inserting or updating operations to EMP_VIEW3 even if EMP_VIEW2 does not include the WITH CHECK OPTION.

The WITH LOCAL CHECK OPTION can also be specified when creating a view. If a view is defined with the LOCAL CHECK OPTION, the definition of the view is used in the checking of any insert or update operations. However, the view does not inherit the search conditions from any updatable views on which it depends.

Deletable views

Depending on how a view is defined, the view can be deletable. A deletable view is a view against which you can successfully issue a DELETE statement.

There are a few rules that must be followed for a view to be considered deletable:

- Each FROM clause of the outer fullselect identifies only one table (with no OUTER clause), deletable view (with no OUTER clause), deletable nested table expression, or deletable common table expression.
- The database manager needs to be able to derive the rows to be deleted in the table using the view definition. Certain operations make this impossible
  - A grouping of multiple rows into one using a GROUP BY clause or column functions result in a loss of the original row and make the view non deletable.
  - Similarly when th rows are derived from a VALUES there is no table to delete from. Again the view is not deletable.
- The outer fullselect doesn't use the GROUP BY or HAVING clauses.
- The outer fullselect doesn't include column functions in its select list.
- The outer fullselect doesn't use set operations (UNION, EXCEPT, or INTERSECT) with the exception of UNION ALL
- The tables in the operands of a UNION ALL must not be the same table, and each operand must be deletable.
- The select list of the outer fullselect does not include DISTINCT.

A view must meet all the rules listed previously to be considered a deletable view. For example, the following view is deletable. It follows all the rules for a deletable view.

```
CREATE VIEW deletable_view
  (number, date, start, end)
AS
  SELECT number, date, start, end
  FROM employee.summary
  WHERE date='01012007'
```

Insertable views

Insertable views allow you to insert rows using the view definition. A view is insertable if an INSTEAD OF trigger for the insert operation has been defined for the view, or at least one column of the view is updatable (independent of an INSTEAD OF trigger for update), and the fullselect of the view does not include UNION ALL. A given row can be inserted into a view (including a UNION ALL) if, and only if, it fulfills the check constraints of exactly one of the underlying tables. To insert into a view that includes non-updatable columns, those columns must be omitted from the column list.

The following example shows an insertable view. However, in this example, an attempt to insert the view will fail. This is because there are columns in the table that do not accept null values. Some of these columns are not present in the view definition. When you try to insert a value using the view, the database manager will try to insert a null value into a NOT NULL column. This action is not permitted.

```
CREATE VIEW insertable_view
  (number, name, quantity)
AS
SELECT number, name, quantify FROM ace.supplies
```

Note: The constraints defined on the table are independent of the operations that can be performed using a view based on that table.

Updatable views

An updatable view is a special case of a deletable view. A deletable view becomes an updatable view when at least one of its columns is updatable.

A column of a view is updatable when all of the following rules are true:

- The view is deletable.
- The column resolves to a column of a table (not using a dereference operation) and the READ ONLY option is not specified.
- All the corresponding columns of the operands of a UNION ALL have exactly matching data types (including length or precision and scale) and matching default values if the fullselect of the view includes a UNION ALL.

The following example uses constant values that cannot be updated. However, the view is a deletable view and at least one of its columns is updatable. Therefore, it is an updatable view.

```
CREATE VIEW updatable_view
  (number, current_date, current_time, temperature)
AS
  SELECT number, CURRENT DATE, CURRENT TIME, temperature)
FROM weather.forecast
WHERE number = 300
```

Read-only views

A view is *read-only* if it is *not* deletable, updatable, or insertable. A view can be read-only if it is a view that does not comply with at least one of the rules for deletable views.

The READONLY column in the SYSCAT.VIEWS catalog view indicates a view is read-only (R).

The following example does not show a deletable view as it uses the DISTINCT clause and the SQL statement involves more than one table:

```
CREATE VIEW read_only_view
  (name, phone, address)
AS
SELECT DISTINCT viewname, viewphone, viewaddress
FROM employee.history adam, employer.dept sales
WHERE adam.id = sales.id
```

Creating views

Views are derived from one or more tables, nicknames, or views, and can be used interchangeably with tables when retrieving data. When changes are made to the data shown in a view, the data is changed in the table itself. The table, nickname, or view on which the view is to be based must already exist before the view can be created.

About this task

A view can be created to limit access to sensitive data, while allowing more general access to other data.

When inserting into a view where the select list of the view definition directly or indirectly includes the name of an identity column of a table, the same rules apply as if the INSERT statement directly referenced the identity column of the table.

In addition to using views as described previously, a view can also be used to:

- Alter a table without affecting application programs. This can happen by creating a view based on an underlying table. Applications that use the underlying table are not affected by the creation of the new view. New applications can use the created view for different purposes than those applications that use the underlying table.

- Sum the values in a column, select the maximum values, or average the values.
- Provide access to information in one or more data sources. You can reference nicknames within the CREATE VIEW statement and create multi-location/global views (the view could join information in multiple data sources located on different systems).

  When you create a view that references nicknames using standard CREATE VIEW syntax, you will see a warning alerting you to the fact that the authentication ID of view users will be used to access the underlying object or objects at data sources instead of the view creator authentication ID. Use the FEDERATED keyword to suppress this warning.

A typed view is based on a predefined structured type. You can create a typed view using the CREATE VIEW statement.

An alternative to creating a view is to use a nested or common table expression to reduce catalog lookup and improve performance.

A sample CREATE VIEW statement is shown in the following example. The underlying table, EMPLOYEE, has columns named SALARY and COMM. For security reasons this view is created from the ID, NAME, DEPT, JOB, and HIREDATE columns. In addition, access on the DEPT column is restricted. This definition will only show the information of employees who belong to the department whose DEPTNO is 10.

```
CREATE VIEW EMP_VIEW1
  (EMPID, EMPNAME, DEPTNO, JOBTITLE, HIREDATE)
  AS SELECT ID, NAME, DEPT, JOB, HIREDATE FROM EMPLOYEE
    WHERE DEPT=10;
```

After the view has been defined, the access privileges can be specified. This provides data security since a restricted view of the table is accessible. As shown in the previous example, a view can contain a WHERE clause to restrict access to certain rows or can contain a subset of the columns to restrict access to certain columns of data.

The column names in the view do not have to match the column names of the base table. The table name has an associated schema as does the view name.

Once the view has been defined, it can be used in statements such as SELECT, INSERT, UPDATE, and DELETE (with restrictions). The DBA can decide to provide a group of users with a higher level privilege on the view than the table.

Creating views that use user-defined functions (UDFs)

Once you create a view that uses a UDF, the view will always use this same UDF as long as the view exists even if you create other UDFs with the same names later. If you want to pick up a new UDF you must re-create the view.

About this task

The following SQL statement creates a view with a function in its definition:

```
CREATE VIEW EMPLOYEE_PENSION (NAME, PENSION)
  AS SELECT NAME, PENSION(HIREDATE,BIRTHDATE,SALARY,BONUS)
  FROM EMPLOYEE
```

The UDF function PENSION calculates the current pension an employee is eligible to receive, based on a formula involving their HIREDATE, BIRTHDATE, SALARY, and BONUS.

Modifying typed views

Certain properties of a typed view can be changed without requiring the view to be dropped and recreated. One such property is the adding of a scope to a reference column of a typed view.

About this task

The ALTER VIEW statement modifies an existing typed view definition by altering a reference type column to add a scope. The DROP statement deletes a typed view. You can also:

- Modify the contents of a typed view through INSTEAD OF triggers
- Alter a typed view to enable statistics collection

Changes you make to the underlying content of a typed view require that you use triggers. Other changes to a typed view require that you drop and then re-create the typed view.

The data type of the column-name in the ALTER VIEW statement must be REF (type of the typed table name or typed view name).

Procedure

To alter a typed view by using the command line, issue the ALTER VIEW statement. For example:

```
ALTER VIEW view_name ALTER column_name
   ADD SCOPE typed_table_or_view_name
```

Results

Other database objects such as tables and indexes are not affected although packages and cached dynamic statements are marked invalid.

Recovering inoperative views

An inoperative view is a view that is no longer available for SQL statements.

About this task

Views can become *inoperative*:

- As a result of a revoked privilege on an underlying table
- If a table, alias, or function is dropped.
- If the superview becomes inoperative. (A superview is a typed view upon which another typed view, a subview, is based.)
- When the views they are dependent on are dropped.

If you do not want to recover an inoperative view, you can explicitly drop it with the DROP VIEW statement, or you can create a view with the same name but a different definition.

An inoperative view has entries only in the SYSCAT.TABLES and SYSCAT.VIEWS catalog views; all entries in the SYSCAT.TABDEP, SYSCAT.TABAUTH, SYSCAT.COLUMNS, and SYSCAT.COLAUTH catalog views are removed.

Procedure

The following steps can help you recover an inoperative view:

1. Determine the SQL statement that was initially used to create the view. You can obtain this information from the TEXT column of the SYSCAT.VIEW catalog view.
2. Set the current schema to the content of the QUALIFIER column.
3. Set the function path to the content of the FUNC_PATH column.
4. Re-create the view by using the CREATE VIEW statement with the same view name and same definition.
5. Use the GRANT statement to regrant all privileges that were previously granted on the view. (Note that all privileges granted on the inoperative view are revoked.)

Dropping views

Use the DROP VIEW statement to drop views. Any views that are dependent on the view being dropped are made inoperative.

Procedure

To drop a view by using the command line, enter:

```
DROP VIEW view_name
```

Example

The following example shows how to drop a view named EMP_VIEW:

```
 DROP VIEW EMP_VIEW
```

As in the case of a table hierarchy, it is possible to drop an entire view hierarchy in one statement by naming the root view of the hierarchy, as in the following example:

```
DROP VIEW HIERARCHY VPerson
```

Chapter 26. Usage lists

A *usage list* is a database object that records each DML statement section that references a particular table or index. A section is the executable form of the query. Statistics are captured for each statement section as it executes. Use usage lists when you want to determine which DML statements, if any, affected a table or index.

Data is collected in a usage list only when the usage list is active. Each entry in a usage list contains data for every DML statement that references the table or index for which the usage list was created. Each entry includes information about the number of times that the section executed and aggregate statistics indicating how the section affected the table or index across all executions.

References in the list can be aggregated by the values that are listed in the following table.

Table 74. Aggregation values for usage list references

Value	Description
executable_ID	Identifies the SQL statement that was executed.
mon_interval_ID	Identifies the monitoring interval at the time that the *executable_ID* was added to the usage list.

Consider the following example of using usage lists. As part of routine monitoring, you see a high value for the **rows_read** monitor element for a specific table in the output for the MON_GET_TABLE table function. You can use a usage list on that table to identify which DML statements contributed to the high value. If you determine that a problem exists, you can use the statistics from the usage list to determine which specific statements might require further monitoring or tuning.

You can create more than one usage list for a table or index. However, activating more than one usage list at a time might negatively affect database performance and memory usage.

Restrictions

The following restrictions apply to usage lists:

- Usage lists can capture information about only DML statements.
- You can create a usage list only for untyped tables. The following table types and objects are not supported:
  - Aliases
  - Created temporary tables
  - Detached tables
  - Hierarchy tables
  - Nicknames
  - Typed tables
  - Views

- You can create a usage list only for the following types of indexes:
  - Block indexes
  - Clustering indexes
  - Dimension block indexes
  - Regular indexes
- The **db2look** utility does not extract the DDL statements that are required to create copies of usage lists.

Usage list memory considerations and validation dependencies

After a usage list is activated, the database manager allocates memory to store the collected data the first time that a section references the object for which the usage list is defined. Throughout the life of the usage list, various actions might affect this memory, invalidate the usage list, or both.

General memory considerations are as follows:

- ***Usage list size considerations:*** Select a reasonable list size or set the **mon_heap_sz** configuration parameter to AUTOMATIC so that the database manager manages the monitor heap size.
- ***Performance considerations:*** To maintain high performance, create usage lists such that they are limited to the amount required to gather the information you need. Each usage list requires system memory; system performance can degrade as additional usage lists are activated.

The following table shows more specifically how various actions affect the allocated memory.

Action	Effect	Effect if usage list is for a partitioned table or index	Effect in a partitioned database environment or DB2 pureScale environment
After you activate a usage list for the first time, a section references the object for which the usage list is defined.	Memory is allocated for the usage list.	Memory is allocated for each data partition. For example, if the usage list requires 2 MB of memory, and three data partitions exist, 6 MB of total memory is allocated.	Memory is allocated for each member. For example, if the usage list requires 2 MB of memory, and three members exist, 6 MB of total memory is allocated.
You change the size of the usage list.	The amount of memory that is associated with the usage list changes the next time that the usage list is activated.	The amount of memory that is associated with the usage list for each data partition changes the next time that the usage list is activated.	The amount of memory that is associated with the usage list for each member changes the next time that the usage list is activated.
You add or attach a new data partition to the table or index for which the usage list is defined.	Does not apply.	Memory is allocated for the new data partition the next time that a section references the table or index.	Does not apply.
You drop the usage list.	The memory that is associated with the usage list is freed.	The memory that is associated with the usage list is freed for all data partitions.	The memory that is associated with the usage list is freed on all members.

Action	Effect	Effect if usage list is for a partitioned table or index	Effect in a partitioned database environment or DB2 pureScale environment
You drop the table or index for which the usage list is defined.	The memory that is associated with the usage list is freed and the catalog entry for the usage list is invalidated. The catalog entry can be validated again by using the ADMIN_REVALIDATE _DB_OBJECTS procedure.	The memory that is associated with the usage list is freed for all data partitions and the catalog entry for the usage list is invalidated. The catalog entry can be validated again by using the ADMIN_REVALIDATE _DB_OBJECTS procedure.	The memory that is associated with the usage list is freed on all members and the catalog entry for the usage list is invalidated. The catalog entry can be validated again by using the ADMIN_REVALIDATE _DB_OBJECTS procedure.
You deactivate the instance or database.	The memory that is associated with the usage list is freed.	The memory that is associated with the usage list is freed for all data partitions.	The memory that is associated with the usage list is freed on all members.
You use the SET USAGE LIST STATE statement to free the memory that is associated with the usage list.	The memory that is associated with the usage list is freed.	The memory that is associated with the usage list is freed for all data partitions.	The memory that is associated with the usage list is freed on all members.
You detach a data partition from the table or index for which the usage list was created.	Does not apply.	The memory that is associated with the data partition that you detached is freed.	Does not apply.
You drop or deactivate a database member.	Does not apply.	Does not apply.	The memory that is associated with the member that you dropped or deactivated is freed.

Chapter 27. pureXML

The pureXML® feature allows you to store well-formed XML documents in database table columns that have the XML data type. By storing XML data in XML columns, the data is kept in its native hierarchical form, rather than stored as text or mapped to a different data model.

Because pureXML data storage is fully integrated, the stored XML data can be accessed and managed by leveraging existing DB2 database server functionality.

The storage of XML data in its native hierarchical form enables efficient search, retrieval, and updates of XML. XQuery, SQL, or a combination of both can be used to query and update XML data. SQL functions that return XML data or take XML arguments (referred to as SQL/XML functions) also enable XML data to be constructed or published from values retrieved from the database.

Querying and updating

XML documents stored in XML columns can be queried and updated using the following methods:

XQuery
XQuery is a generalized language for interpreting, retrieving, and modifying XML data. The DB2 database server allows XQuery to be invoked directly or from within SQL. Because the XML data is stored in DB2 tables and views, functions are provided that extract the XML data from specified tables and views by naming the table or view directly, or by specifying an SQL query. XQuery supports various expressions for processing XML data, for updating existing XML objects such as elements and attributes, and for constructing new XML objects. The programming interface to XQuery provides facilities similar to those of SQL to execute queries and retrieve results.

SQL statements and SQL/XML functions
Many SQL statements support the XML data type. This enables you to perform many common database operations with XML data, such as creating tables with XML columns, adding XML columns to existing tables, creating indexes over XML columns, creating triggers on tables with XML columns, and inserting, updating, or deleting XML documents. The set of SQL/XML functions, expressions, and specifications supported by DB2 database server has been enhanced to take full advantage of the XML data type.

XQuery can be invoked from within an SQL query. In this case, the SQL query can pass data to XQuery in the form of bound variables.

Application development

Support for application development is provided by several programming languages, and through SQL and external procedures:

Programming language support
Application development support of the new pureXML feature enables applications to combine XML and relational data access and storage. The following programming languages support the XML data type:

- C or C++ (embedded SQL or CLI)
- COBOL
- Java (JDBC or SQLJ)
- C# and Visual Basic (IBM Data Server Provider for .NET)
- PHP
- Perl

SQL and external procedures
XML data can be passed to SQL procedures and external procedures by including parameters of data type XML in CREATE PROCEDURE parameter signatures. Existing procedure features support the implementation of procedural logic flow around SQL statements that produce or make use of XML values as well as the temporary storage of XML data values in variables.

Administration

The pureXML feature provides a repository for managing the URI dependencies of XML documents and enables XML data movement for database administration:

XML schema repository (XSR)
The XML schema repository (XSR) is a repository for all XML artifacts required to process XML instance documents stored in XML columns. It stores XML schemas, DTDs, and external entities referenced in XML documents.

Import, export and load utilities
The import, export and load utilities have been updated to support the native XML data type. These utilities treat XML data like LOB data: both types of data are stored outside the actual table. Application development support for importing, exporting and loading XML data is also provided by updated db2Import, db2Export and db2Load APIs. These updated utilities permit data movement of XML documents stored in XML columns that is similar to the data movement support for relational data.

Performance

Several performance oriented features are available to you when working with XML documents stored in XML columns:

Indexes over XML data
Indexing support is available for data stored in XML columns. The use of indexes over XML data can improve the efficiency of queries issued against XML documents. Similar to a relational index, an index over XML data indexes a column. They differ, however, in that a relational index indexes an entire column, while an index over XML data indexes part of a column. You indicate which parts of an XML column are indexed by specifying an XML pattern, which is a limited XPath expression.

Optimizer
The optimizer has been updated to support the evaluation of SQL, XQuery, and SQL/XML functions that embed XQuery, against XML and relational data. The optimizer exploits statistics gathered over XML data, as well as data from indexes over XML data, to produce efficient query execution plans.

Explain facility
The Explain facility has been updated to support SQL enhancements for

querying XML data and to support XQuery expressions. These updates to the Explain facility allow you to see quickly how DB2 database server evaluates query statements against XML data.

Tooling

Support for the XML data type is available in tools including the command line processor, IBM Data Studio, and IBM Database Add-Ins for Microsoft Visual Studio.

Annotated XML schema decomposition

The pureXML feature enables you to store and access XML data as XML, in its hierarchical form, there can be cases where accessing XML data as relational data is required. Annotated XML schema decomposition decomposes documents based on annotations specified in an XML schema.

Comparison of the XML model and the relational model

When you design your databases, you must decide whether your data is better suited to the XML model or the relational model. Take advantage of the hybrid nature of DB2 databases that supports both relational and XML data in a single database.

While this discussion explains some of the main differences between the models and the factors that apply to each, there are numerous factors that can determine the most suitable choice for your implementation. Use this discussion as a guideline to assess the factors that can impact your specific implementation.

Major differences between XML data and relational data

XML data is hierarchical; relational data is represented in a model of logical relationships

An XML document contains information about the relationship of data items to each other in the form of the hierarchy. With the relational model, the only types of relationships that can be defined are parent table and dependent table relationships.

XML data is self-describing; relational data is not

An XML document contains not only the data, but also tagging for the data that explains what it is. A single document can have different types of data. With the relational model, the content of the data is defined by its column definition. All data in a column must have the same type of data.

XML data has inherent ordering; relational data does not

For an XML document, the order in which data items are specified is assumed to be the order of the data in the document. There is often no other way to specify order within the document. For relational data, the order of the rows is not guaranteed unless you specify an ORDER BY clause on one or more columns.

Factors influencing data model choice

What kind of data you store can help you determine how you store it. For example, if the data is naturally hierarchical and self-describing, you might store it as XML data. However, there are other factors that might influence your decision about which model to use:

When you need maximum flexibility

Relational tables follow a fairly rigid model. For example, normalizing one table into many or denormalizing many tables into one can be very difficult. If the data design changes often, representing it as XML data is a better choice. XML schemas can be evolved over time, for example.

When you need maximum performance for data retrieval

Some expense is associated with serializing and interpreting XML data. If performance is more of an issue than flexibility, relational data might be the better choice.

When data is processed later as relational data

If subsequent processing of the data depends on the data being stored in a relational database, it might be appropriate to store parts of the data as relational, using decomposition. An example of this situation is when online analytical processing (OLAP) is applied to the data in a data warehouse. Also, if other processing is required on the XML document as a whole, then storing some of the data as relational as well as storing the entire XML document might be a suitable approach in this case.

When data components have meaning outside a hierarchy

Data might be inherently hierarchical in nature, but the child components do not need the parents to provide value. For example, a purchase order might contain part numbers. The purchase orders with the part numbers might be best represented as XML documents. However, each part number has a part description associated with it. It might be better to include the part descriptions in a relational table, because the relationship between the part numbers and the part descriptions is logically independent of the purchase orders in which the part numbers are used.

When data attributes apply to all data, or to only a small subset of the data

Some sets of data have a large number of possible attributes, but only a small number of those attributes apply to any particular data value. For example, in a retail catalog, there are many possible data attributes, such as size, color, weight, material, style, weave, power requirements, or fuel requirements. For any given item in the catalog, only a subset of those attributes is relevant: power requirements are meaningful for a table saw, but not for a coat. This type of data is difficult to represent and search with a relational model, but relatively easy to represent and search with an XML model.

When the ratio of data complexity to volume is high

Many situations involve highly structured information in very small quantities. Representation of that data with a relational model can involve complex star schemas in which each dimension table is joined to many more dimension tables, and most of the tables have only a few rows. A better way to represent this data is to use a single table with an XML column, and to create views on that table, where each view represents a dimension.

When referential integrity is required

XML columns cannot be defined as part of referential constraints. Therefore, if values in XML documents need to participate in referential constraints, you should store the data as relational data.

When the data needs to be updated often

You update XML data in an XML column only by replacing full documents. If you need to frequently update small fragments of very large documents for a large number of rows, it can be more efficient to store the data in non-XML columns. If, however, you are updating small documents and only a few documents at a time, storing as XML can be efficient as well.

XML data type

Use the XML data type to define columns of a table and store XML values. All XML values must be well-formed XML documents. You can use this native data type to store well-formed XML documents in their native hierarchical format in the database alongside other relational data.

XML values are processed in an internal representation that is not a string and not directly comparable to string values. An XML value can be transformed into a serialized string value representing the XML document using the XMLSERIALIZE function or by binding the value to an application variable of an XML, string, or binary type. Similarly, a string value that represents an XML document can be transformed to an XML value using the XMLPARSE function or by binding an application string, binary, or XML application type to an XML value. In SQL data change statements (such as INSERT) involving XML columns, a string or binary value that represents an XML document is transformed into an XML value using an injected XMLPARSE function. An XML value can be implicitly parsed or serialized when exchanged with application string and binary data types.

There is no architectural limit on the size of an XML value in a database. However, note that serialized XML data exchanged with DB2 database server is effectively limited to 2 GB.

XML documents can be inserted, updated and deleted using SQL data manipulation statements. Validation of an XML document against an XML schema, typically performed during insert or update, is supported by the XML schema repository (XSR). The DB2 database system also provides mechanisms for constructing and querying XML values, as well as exporting and importing XML data. An index over XML data can be defined on an XML column, providing improved search performance of XML data. The XML data in table or view columns can be retrieved as serialized string data through various application interfaces.

Creation of tables with XML columns

To create tables with XML columns, you specify columns with the XML data type in the CREATE TABLE statement. A table can have one or more XML columns.

You do not specify a length when you define an XML column. However, serialized XML data that is exchanged with a DB2 database is limited to 2 GB per value of type XML, so the effective limit of an XML document is 2 GB.

Like a LOB column, an XML column holds only a descriptor of the column. The data is stored separately.

Note:

- If you enable data row compression for the table, XML documents require less storage space.
- You can optionally store smaller and medium-size XML documents in the row of the base table instead of storing them in the default XML storage object.

Example: The sample database contains a table for customer data that contains two XML columns. The definition looks like this:

```
CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY,
                       Info XML,
                       History XML)
```

Example: The VALIDATED predicate checks whether the value in the specified XML column has been validated. You can define a table check constraint on XML columns, using the VALIDATED predicate, to ensure that all documents inserted or updated in a table are valid.

```
CREATE TABLE TableValid (id BIGINT,
                         xmlcol XML,
                         CONSTRAINT valid_check CHECK (xmlcol IS VALIDATED))
```

Example: Setting the COMPRESS attribute to YES enables data row compression. XML documents stored in XML columns are subject to row compression. Compressing data at the row level allows repeating patterns to be replaced with shorter symbol strings.

```
CREATE TABLE TableXmlCol (id BIGINT,
                         xmlcol XML) COMPRESS YES
```

Example: The following CREATE TABLE statement creates a patient table partitioned by visit date. All records between January 01, 2000 and December 31, 2006 are in the first partition. The more recent data are partitioned every 6 months.

```
CREATE TABLE Patients ( patientID BIGINT, visit_date DATE, diagInfo XML,
    prescription XML )
    INDEX IN indexTbsp LONG IN ltbsp
    PARTITION BY ( visit_date )
       ( STARTING '1/1/2000' ENDING '12/31/2006',
         STARTING '1/1/2007' ENDING '6/30/2007',
            ENDING '12/31/2007',
            ENDING '6/30/2008',
            ENDING '12/31/2008',
            ENDING '6/30/2009' );
```

Addition of XML columns to existing tables

To add XML columns to existing tables, you specify columns with the XML data type in the ALTER TABLE statement with the ADD clause. A table can have one or more XML columns.

Example The sample database contains a table for customer data that contains two XML columns. The definition looks like this:

```
CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY,
                       Info XML,
                       History XML)
```

Create a table named MyCustomer that is a copy of Customer, and add an XML column to describe customer preferences:

```
CREATE TABLE MyCustomer LIKE Customer;
ALTER TABLE MyCustomer ADD COLUMN Preferences XML;
```

Example: Setting the COMPRESS attribute to YES enables data row compression. XML documents stored in XML columns are subject to row compression. Compressing data at the row level allows repeating patterns to be replaced with shorter symbol strings.

```
ALTER TABLE MyCustomer ADD COLUMN Preferences XML COMPRESS YES;
```

Example: The following CREATE TABLE statement creates a patient table partitioned by visit date. All records between January 01, 2000 and December 31, 2006 are in the first partition. The more recent data are partitioned every 6 months.

```
CREATE TABLE Patients ( patientID INT, Name Varchar(20), visit_date DATE,
      diagInfo XML )
    PARTITION BY ( visit_date )
       ( STARTING '1/1/2000' ENDING '12/31/2006',
         STARTING '1/1/2007' ENDING '6/30/2007',
            ENDING '12/31/2007',
            ENDING '6/30/2008',
            ENDING '12/31/2008',
            ENDING '6/30/2009' );
```

The following ALTER table statement adds another XML column for patient prescription information:

```
ALTER TABLE Patients ADD COLUMN prescription XML ;
```

Inserting XML columns

To insert data into an XML column, use the SQL INSERT statement. The input to the XML column must be a well-formed XML document, as defined in the XML 1.0 specification. The application data type can be an XML, character, or binary type.

It is recommended that XML data be inserted from host variables, rather than literals, so that the DB2 database server can use the host variable data type to determine some of the encoding information.

XML data in an application is in its serialized string format. When you insert the data into an XML column, it must be converted to its XML hierarchical format. If the application data type is an XML data type, the DB2 database server performs this operation implicitly. If the application data type is not an XML type, you can invoke the XMLPARSE function explicitly when you perform the insert operation, to convert the data from its serialized string format to the XML hierarchical format.

During document insertion, you might also want to validate the XML document against a registered XML schema. You can do that with the XMLVALIDATE function.

The following examples demonstrate how XML data can be inserted into XML columns. The examples use table MyCustomer, which is a copy of the sample Customer table. The XML data that is to be inserted is in file c6.xml, and looks like this:

```
<customerinfo Cid="1015">
  <name>Christine Haas</name>
  <addr country="Canada">
    <street>12 Topgrove</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X-7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-5238</phone>
  <phone type="home">416-555-2934</phone>
</customerinfo>
```

Example: In a JDBC application, read XML data from file c6.xml as binary data, and insert the data into an XML column:

```
PreparedStatement insertStmt = null;
String sqls = null;
int cid = 1015;
```

```
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
File file = new File("c6.xml");
insertStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
insertStmt.executeUpdate();
```

Example: In a static embedded C application, insert data from a binary XML host variable into an XML column:

```
EXEC SQL BEGIN DECLARE SECTION;
   sqlint64 cid;
   SQL TYPE IS XML AS BLOB (10K) xml_hostvar;
EXEC SQL END DECLARE SECTION;
...
cid=1015;
/* Read data from file c6.xml into xml_hostvar */
...
EXEC SQL INSERT INTO MyCustomer (Cid,Info) VALUES (:cid, :xml_hostvar);
```

Querying XML data

You can query or retrieve XML data stored in the database through two main query languages, either by using each language on its own or by using a combination of the two.

The following options are available to you:

- XQuery expressions only
- XQuery expressions that invoke SQL statements
- SQL statements only
- SQL statements that executes XQuery expressions

These various methods allow you to query or retrieve XML and other relational data from either an SQL or XQuery context.

Pieces of or entire XML documents can be queried and retrieved using these methods. Queries can return fragments or entire XML documents, and results returned from queries can be limited by using predicates. Because queries on XML data return XML sequences, a query's result can be used in the construction of XML data as well.

Comparison of methods for querying XML data

Because XML data can be queried in a number of ways, using XQuery, SQL, or a combination of these, the method to choose can differ depending on your situation. The following sections describe conditions that are advantageous for a particular query method.

XQuery only

Querying with XQuery alone can be a suitable choice when:

- applications access only XML data, without the need to query non-XML relational data
- migrating queries previously written in XQuery to DB2 for Linux, UNIX, and Windows
- returning query results to be used as values for constructing XML documents
- the query author is more familiar with XQuery than SQL

XQuery that invokes SQL

Querying with XQuery that invokes SQL can be a suitable choice when (in addition to the scenarios identified in the previous section on using XQuery only):

- queries involve XML data and relational data; SQL predicates and indexes defined on the relational columns can be leveraged in the query
- you want to apply XQuery expressions to the results of:
  - UDF calls, as these cannot be invoked directly from XQuery
  - XML values constructed from relational data using SQL/XML publishing functions
  - queries that use DB2 Net Search Extender which offers full text search of XML documents but which must be used with SQL

SQL only

When retrieving XML data using only SQL, without any XQuery, you can query only at the XML column level. For this reason, only entire XML documents can be returned from the query. This usage is suitable when:

- you want to retrieve entire XML documents
- you do not need to query based on values within the stored documents, or where the predicates of your query are on other non-XML columns of the table

SQL/XML functions that execute XQuery expressions

The SQL/XML functions XMLQUERY and XMLTABLE, as well as the XMLEXISTS predicate, enable XQuery expressions to be executed from within the SQL context. Executing XQuery within SQL can be a suitable choice when:

- existing SQL applications need to be enabled for querying within XML documents. To query within XML documents, XQuery expressions need to be executed, which can be done using SQL/XML
- applications querying XML data need to pass parameter markers to the XQuery expression. (The parameter markers are first bound to XQuery variables in XMLQUERY or XMLTABLE.)
- the query author is more familiar with SQL than XQuery
- both relational and XML data needs to be returned in a single query
- you need to join XML and relational data
- you want to group or aggregate XML data. You can apply the GROUP BY or ORDER BY clauses of a subselect to the XML data (for example, after the XML data has been retrieved and collected in table format by using the XMLTABLE function)

Indexing XML data

An index over XML data can be used to improve the efficiency of queries on XML documents that are stored in an XML column.

In contrast to traditional relational indexes, where index keys are composed of one or more table columns you specify, an index over XML data uses a particular XML pattern expression to index paths and values in XML documents stored within a single column. The data type of that column must be XML.

Instead of providing access to the beginning of a document, index entries in an index over XML data provide access to nodes within the document by creating index keys based on XML pattern expressions. Because multiple parts of a XML document can satisfy an XML pattern, multiple index keys may be inserted into the index for a single document.

You create an index over XML data using the CREATE INDEX statement, and drop an index over XML data using the DROP INDEX statement. The GENERATE KEY USING XMLPATTERN clause you include with the CREATE INDEX statement specifies what you want to index.

Some of the keywords used with the CREATE INDEX statement for indexes on non-XML columns do not apply to indexes over XML data. The UNIQUE keyword also has a different meaning for indexes over XML data.

Example: Creating an index over XML data

Suppose that table companyinfo has an XML column named companydocs, which contains XML document fragments like these:

Document for Company1

```
<company name="Company1">
  <emp id="31201" salary="60000" gender="Female">
    <name>
      <first>Laura</first>
      <last>Brown</last>
    </name>
    <dept id="M25">
      Finance
    </dept>
  </emp>
</company>
```

Document for Company2

```
<company name="Company2">
  <emp id="31664" salary="60000" gender="Male">
    <name>
      <first>Chris</first>
      <last>Murphy</last>
    </name>
    <dept id="M55">
      Marketing
    </dept>
  </emp>
  <emp id="42366" salary="50000" gender="Female">
    <name>
      <first>Nicole</first>
      <last>Murphy</last>
    </name>
    <dept id="K55">
      Sales
    </dept>
  </emp>
</company>
```

Users of the companyinfo table often retrieve employee information using the employee ID. You might use an index like this one to make that retrieval more efficient:

```
CREATE INDEX empindex on companyinfo(companydocs)        1
  GENERATE KEY USING XMLPATTERN '/company/emp/@id'       2
   AS SQL DOUBLE                                     3
```

Figure 36. Example of an index over XML data

Notes to Figure 36 on page 404:

1 The index over XML data is defined on the companydocs column of the companyinfo table. companydocs must be of the XML data type.

2 The GENERATE KEY USING XMLPATTERN clause provides information about what you want to index. This clause is called an XML index specification. The XML index specification contains an XML pattern clause. The XML pattern clause in this example indicates that you want to index the values of the id attribute of each employee element.

3 AS SQL DOUBLE indicates that indexed values are stored as DOUBLE values.

Updating XML data

To update data in an XML column, use the SQL UPDATE statement. Include a WHERE clause when you want to update specific rows.

The entire column value will be replaced. The input to the XML column must be a well-formed XML document. The application data type can be an XML, character, or binary type.

When you update an XML column, you might also want to validate the input XML document against a registered XML schema. You can do that with the XMLVALIDATE function.

You can use XML column values to specify which rows are to be updated. To find values within XML documents, you need to use XQuery expressions. One way of specifying XQuery expressions is the XMLEXISTS predicate, which allows you to specify an XQuery expression and determine if the expression results in an empty sequence. When XMLEXISTS is specified in the WHERE clause, rows will be updated if the XQuery expression returns a non-empty sequence.

The following examples demonstrate how XML data can be updated in XML columns. The examples use table MYCUSTOMER, which is a copy of the sample CUSTOMER table. The examples assume that MYCUSTOMER already contains a row with a customer ID value of 1004. The XML data that updates existing column data is assumed to be stored in a file c7.xml, whose contents look like this:

```
<customerinfo Cid="1004">
  <name>Christine Haas</name>
  <addr country="Canada">
    <street>12 Topgrove</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9Y-8G9</pcode-zip>
  </addr>
  <phone type="work">905-555-5238</phone>
  <phone type="home">416-555-2934</phone>
</customerinfo>
```

Example: In a JDBC application, read XML data from file c7.xml as binary data, and use it to update the data in an XML column:

```
PreparedStatement updateStmt = null;
String sqls = null;
int cid = 1004;
sqls = "UPDATE MyCustomer SET Info=? WHERE Cid=?";
updateStmt = conn.prepareStatement(sqls);
```

```
updateStmt.setInt(1, cid);
File file = new File("c7.xml");
updateStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
updateStmt.executeUpdate();
```

Example: In an embedded C application, update data in an XML column from a binary XML host variable:

```
EXEC SQL BEGIN DECLARE SECTION;
   sqlint64 cid;
   SQL TYPE IS XML AS BLOB (10K) xml_hostvar;
EXEC SQL END DECLARE SECTION;
...
cid=1004;
/* Read data from file c7.xml into xml_hostvar */
...
EXEC SQL UPDATE MyCustomer SET Info=:xml_hostvar WHERE Cid=:cid;
```

In these examples, the value of the Cid attribute within the <customerinfo> element happens to be stored in the CID relational column as well. Because of this, the WHERE clause in the UPDATE statements used the relational column CID to specify the rows to update. In the case where the values that determine which rows are chosen for update are found only within the XML documents themselves, the XMLEXISTS predicate can be used. For example, the UPDATE statement in the previous embedded C application example can be changed to use XMLEXISTS as follows:

```
EXEC SQL UPDATE MyCustomer SET Info=:xml_hostvar
       WHERE XMLEXISTS ('$doc/customerinfo[@Cid = $c]'
                        passing INFO as "doc", cast(:cid as integer) as "c");
```

Example: The following example updates existing XML data from the MYCUSTOMER table. The SQL UPDATE statement operates on a row of the MYCUSTOMER table and replaces the document in the INFO column of the row with the logical snapshot of the document modified by the transform expression:

```
UPDATE MyCustomer
SET info = XMLQUERY(
   'transform
    copy $newinfo := $info
    modify do insert <status>Current</status>
      as last into $newinfo/customerinfo
    return $newinfo' passing info as "info")
WHERE cid = 1004
```

XML data movement

Support for XML data movement is provided by the load, import and export utilities. Support for moving tables that contain XML columns without taking the tables offline is provided by the ADMIN_MOVE_TABLE stored procedure.

Importing XML data

The import utility can be used to insert XML documents into a regular relational table. Only well-formed XML documents can be imported.

Use the XML FROM option of the IMPORT command to specify the location of the XML documents to import. The XMLVALIDATE option specifies how imported documents should be validated. You can select to have the imported XML data validated against a schema specified with the IMPORT command, against a schema identified by a schema location hint inside of the source XML document, or by the schema identified by the XML Data Specifier in the main data file. You can also use the XMLPARSE option to specify how whitespace is handled when the XML document is imported. The `xmlchar` and `xmlgraphic` file type modifiers allow you to specify the encoding characteristics for the imported XML data.

Loading XML data

The load utility offers an efficient way to insert large volumes of XML data into a table. This utility also allows certain options unavailable with the import utility, such as the ability to load from a user-defined cursor.

Like the IMPORT command, with the LOAD command you can specify the location of the XML data to load, validation options for the XML data, and how whitespace is handled. As with IMPORT, you can use the `xmlchar` and `xmlgraphic` file type modifiers to specify the encoding characteristics for the loaded XML data.

Exporting XML data

Data may be exported from tables that include one or more columns with an XML data type. Exported XML data is stored in files separate from the main data file containing the exported relational data. Information about each exported XML document is represented in the main exported data file by an XML data specifier (XDS). The XDS is a string that specifies the name of the system file in which the XML document is stored, the exact location and length of the XML document inside of this file, and the XML schema used to validate the XML document.

You can use the XMLFILE, XML TO, and XMLSAVESCHEMA parameters of the EXPORT command to specify details about how exported XML documents are stored. The `xmlinsepfiles`, `xmlnodeclaration`, `xmlchar`, and `xmlgraphic` file type modifiers allow you to specify further details about the storage location and the encoding of the exported XML data.

Moving tables online

The ADMIN_MOVE_TABLE stored procedure moves the data in an active table into a new table object with the same name, while the data remains online and available for access. The table can include one or more columns with an XML data type. Use an online table move instead of an offline table move if you value availability more than cost, space, move performance, and transaction overhead

You can call the procedure once or multiple times, one call for each operation performed by the procedure. Using multiple calls provides you with additional options, such as cancelling the move or controlling when the target table is taken offline to be updated.

pureXML tutorial

You can use this tutorial to learn how to set up a DB2 database to store XML data and to perform basic operations with the pureXML feature. The pureXML XML data type can store a single, well-formed XML document in each row. The tutorial is available at http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.xml.doc/doc/c0023610.html.

Part 4. Monitoring DB2 Activity

Monitoring DB2 Activity consist of performing tasks that are associated with examining the operational status of your database. DB2 provides multiples interfaces for database and workload monitoring. It also provides tools for obtaining information about access plans and troubleshooting problems.

Database monitoring is a vital activity for the maintenance of the performance and health of your database management system. To facilitate monitoring, DB2 collects information from the database manager, its databases, and any connected applications. With this information you can perform the following types of tasks, and more:

- Forecast hardware requirements based on database usage patterns.
- Analyze the performance of individual applications or SQL queries.
- Track the usage of indexes and tables.
- Pinpoint the cause of poor system performance.
- Assess the impact of optimization activities (for example, altering database manager configuration parameters, adding indexes, or modifying SQL queries).

Chapter 28. Database monitoring

There are two ways to monitor operations in your database. You can view information that shows the state of various aspects of the database at a specific point in time. Or, you can set up event monitors to capture historical information as specific types of database events take place.

You can monitor your database operations in real-time using monitoring table functions. For example, you can use a monitoring table function to examine the total amount of space used in a table space. These table functions let you examine *monitor elements* and metrics that report on virtually all aspects of database operations using SQL. The monitoring table functions use the newer, lightweight, high-speed monitoring infrastructure that was introduced in Version 9.7. In addition to the table functions, snapshot monitoring routines are also available. The snapshot monitoring facilities in DB2 use monitoring infrastructure that existed before Version 9.7. Generally speaking, snapshot monitoring facilities are no longer being enhanced in the product; where possible, use the monitoring table functions to retrieve the data you want to see.

Event monitors capture information about database operations over time, as specific types of events occur. For example, you can create an event monitor to capture information about locks and deadlocks as they occur in the system. Or you might create an event monitor to record when a threshold that you specify (for example the total processor time used by an application or workload) is exceeded. Event monitors generate output in different formats; all of them can write event data to regular tables; some event monitors have additional output options.

IBM InfoSphere Optim Performance Manager provides a Web interface that you can use to isolate and analyze typical database performance problems. You can also view a summary of the health of your databases and drill down. For more details, see Monitoring with Optim Performance Manager at http://publib.boulder.ibm.com/infocenter/idm/docv3/topic/com.ibm.datatools.perfmgmt.monitor.doc/p_monitor.html.

Monitoring DB2 Activity with table functions

Starting with DB2 Version 9.7, you can access monitor data through a light-weight alternative to the traditional system monitor. Use monitor table functions to collect and view data for systems, activities, or data objects.

Data for monitored elements are continually accumulated in memory and available for querying. You can choose to receive data for a single object (for example, service class A or table TABLE1) or for all objects.

When using these table functions in a database partitioned environment, you can choose to receive data for a single partition or for all partitions. If you choose to receive data for all partitions, the table functions return one row for each partition. Using SQL, you can sum the values across partitions to obtain the value of a monitor element across partitions.

Monitoring system information using table functions

The system monitoring perspective encompasses the complete volume of work and effort expended by the data server to process application requests. From this perspective, you can determine what the data server is doing as a whole as well as for particular subsets of application requests.

Monitor elements for this perspective, referred to as request monitor elements, cover the entire range of data server operations associated with processing requests.

Request monitor elements are continually accumulated and aggregated in memory so they are immediately available for querying. Request monitor elements are aggregated across requests at various levels of the workload management (WLM) object hierarchy: by unit of work, by workload, by service class. They are also aggregated by connection.

Use the following table functions for accessing current system monitoring information:

- MON_GET_SERVICE_SUBCLASS and MON_GET_SERVICE_SUBCLASS_DETAILS
- MON_GET_WORKLOAD and MON_GET_WORKLOAD_DETAILS
- MON_GET_CONNECTION and MON_GET_CONNECTION_DETAILS
- MON_GET_UNIT_OF_WORK and MON_GET_UNIT_OF_WORK_DETAILS

This set of table functions enables you to drill down or focus on request monitor elements at a particular level of aggregation. Table functions are provided in pairs: one for relational access to commonly used data and the other for XML access to the complete set of available monitor elements.

The system monitoring information is collected by these table functions by default for a new database. You can change default settings using one or both of the following settings:

- The database configuration parameter **mon_req_metrics** specifies the minimum level of collection in all service classes.
- The COLLECT REQUEST METRICS clause of the CREATE/ALTER SERVICE CLASS statement specifies the level of collection for a service superclass. Use this setting to increase the level of collection for a given service class over the minimum level of collection set for all service classes.

The possible values for each setting are the following:

None No request monitor elements are collected

Base All request monitor elements are collected

For example, to collect system monitoring information for only a subset of service classes, do the following:

1. Set the database configuration parameter **mon_req_metrics** to NONE.
2. For each required service class, set the COLLECT REQUEST METRICS clause of the CREATE/ALTER SERVICE CLASS statement to BASE.

Monitoring activities using table functions

The activity monitoring perspective focuses on the subset of data server processing related to executing activities. In the context of SQL statements, the term activity refers to the execution of the section for a SQL statement.

Monitor elements for this perspective, referred to as activity monitor elements, are a subset of the request monitor elements. Activity monitor elements measure aspects of work done for statement section execution. Activity monitoring includes other information such as SQL statement text for the activity.

For activities in progress, activity metrics are accumulated in memory. For activities that are SQL statements, activity metrics are also accumulated in the package cache. In the package cache activity metrics are aggregated over all executions of each SQL statement section.

Use the following table functions to access current data for activities:

MON_GET_ACTIVITY_DETAILS
Returns data about the individual activities in progress when the table function is called. Data is returned in a relational form, however, the detailed metrics are returned in an XML document in the DETAILS column of the results table.

MON_GET_PKG_CACHE_STMT
Returns a point-in-time view of both static and dynamic SQL statements in the database package cache. Data is returned in a relational form.

MON_GET_PKG_CACHE_STMT_DETAILS
Returns detailed metrics for one or more package cache entries. Data is returned in a relational form, however, the detailed metrics are returned in an XML document in the DETAILS column of the results table.

Activity monitoring information is collected by default for a new database. You can change default settings using one or both of the following settings:

- The **mon_act_metrics** database configuration parameter specifies the minimum level of collection in all workloads.
- The COLLECT ACTIVITY METRICS clause of the CREATE/ALTER WORKLOAD statement specifies the level of collection for a given workload over the minimum level of collection set for all workloads.

The possible values for each setting are the following:

None No activity monitor elements are collected

Base All activity monitor elements are collected

For example, to collect activity monitor elements for only selected workloads, do the following:

1. Set the **mon_act_metrics** database configuration parameter to NONE.
2. Set the COLLECT ACTIVITY METRICS clause of the CREATE/ALTER WORKLOAD statement to BASE. By default, the values for other workloads is NONE.

Monitoring data objects using table functions

The data object monitoring perspective provides information about operations performed on data objects, that is tables, indexes, buffer pools, table spaces, and containers.

A different set of monitor elements is available for each object type. Monitor elements for a data object are incremented each time a request involves processing that object. For example, when processing a request that involves reading rows from a particular table, the metric for rows read is incremented for that table.

Use the following table functions to access current details for data objects:

- MON_GET_BUFFERPOOL
- MON_GET_TABLESPACE
- MON_GET_CONTAINER
- MON_GET_TABLE
- MON_GET_INDEX

These table functions return data in a relational form.

You cannot access historical data for data objects.

Data object monitor elements are collected by default for new databases. You can use the **mon_obj_metrics** database configuration parameter to reduce the amount of data collected by the table functions.

The possible values for this configuration parameter are the following:

None No data object monitor elements are collected

Base Some data object monitor elements are collected

Extended
All data object monitor elements are collected

To stop collecting data object monitor elements reported by the following table functions, set the **mon_obj_metrics** configuration parameter to NONE.

- MON_GET_BUFFERPOOL
- MON_GET_TABLESPACE
- MON_GET_CONTAINER

Object usage

When SQL statements are executed, they use various database objects, such as tables and indexes. Knowing which database objects a statement accesses and how the statement affects them can help you identify targets for monitoring or performance tuning.

The following table shows the entities that you can use to explore the relationship between database objects and statements.

Table 75. Ways to identify object usage

Mechanism	Definition	Usage
Usage list	A usage list is a database object that records each DML statement section that references a particular table or index and captures statistics about that section as it executes.	Identify the statements that affected a table or index. If you notice an unusual value for a metric when monitoring a database object, use a usage list to determine whether a particular statement contributed to that metric. You can also view statistics for each statement that affected the object.

Table 75. Ways to identify object usage (continued)

Mechanism	Definition	Usage
Section explain with actuals	A section explain is a set of information about the access plan that the optimizer chose for an SQL statement. You can capture section actuals as part of the explain. Section actuals are runtime statistics that are collected when a section executes.	Identify the tables or indexes that a statement affects. You can view statistics for each table or index and use these statistics to determine how the statement affects each object and where tuning might be required.

You can use the information in a usage list or section explain with actuals as baseline data for performance tuning. Collect information about object usage before tuning statements or database configuration parameters. After tuning, collect the information again to verify that tuning improved performance.

Identifying the statements that affect a table

Use usage lists to identify DML statement sections that affect a particular table when the statement sections execute. You can view statistics for each statement and use these statistics to determine where additional monitoring or tuning might be required.

Before you begin

Do the following tasks:

- Identify a table for which you want to view object usage statistics. You can use the MON_GET_TABLE table function to view monitor metrics for one or more tables.
- To issue the required statements, ensure that the privileges that are held by the authorization ID of each statement include DBADM authority or SQLADM authority.
- Ensure that you have EXECUTE privilege on the MON_GET_TABLE_USAGE_LIST and MON_GET_USAGE_LIST_STATUS table functions.

About this task

When you view the output of the MON_GET_TABLE table function, you might see an unusual value for a monitor element. You can use usage lists to determine whether any DML statements contributed to this value.

Usage lists contain statistics about factors like locks and buffer pool usage for each statement that affected a table during a particular time frame. If you determine that a statement affected a table negatively, use these statistics to determine where further monitoring might be required or how the statement can be tuned.

Procedure

To identify the statements that affect a table:

1. Set the **mon_obj_metrics** configuration parameter to EXTENDED by issuing the following command:

```
DB2 UPDATE DATABASE CONFIGURATION USING MON_OBJ_METRICS EXTENDED
```

Setting this configuration parameter to EXTENDED ensures that statistics are collected for each entry in the usage list.

2. Create a usage list for the table by using the CREATE USAGE LIST statement. For example, to create the INVENTORYUL usage list for the SALES.INVENTORY table, issue the following command:

```
CREATE USAGE LIST INVENTORYUL FOR TABLE SALES.INVENTORY
```

3. Activate the collection of object usage statistics by using the SET USAGE LIST STATE statement. For example, to activate collection for the INVENTORYUL usage list, issue the following command:

```
SET USAGE LIST INVENTORYUL STATE = ACTIVE
```

4. During the collection of object statistics, ensure that the usage list is active and that sufficient memory is allocated for the usage list by using the MON_GET_USAGE_LIST_STATUS table function. For example, to check the status of the INVENTORYUL usage list, issue the following command:

```
SELECT MEMBER,
       STATE,
       LIST_SIZE,
       USED_ENTRIES,
       WRAPPED
FROM TABLE(MON_GET_USAGE_LIST_STATUS('SALES', 'INVENTORYUL', -2))
```

5. When the time period for which you want to collect object usage statistics is elapsed, deactivate the collection of usage list data by using the SET USAGE LIST STATE statement. For example, to deactivate collection for the INVENTORYUL usage list, issue the following command:

```
SET USAGE LIST SALES.INVENTORYUL STATE = INACTIVE
```

6. View the information that you collected by using the MON_GET_TABLE_USAGE_LIST function. You can view statistics for a subset or for all of the statements that affected the table during the time period for which you collected statistics. For example, to see only the 10 statements that read the most rows of the table, issue the following command:

```
SELECT MEMBER,
       EXECUTABLE_ID,
       NUM_REFERENCES,
       NUM_REF_WITH_METRICS,
       ROWS_READ,
       ROWS_INSERTED,
       ROWS_UPDATED,
       ROWS_DELETED
FROM TABLE(MON_GET_TABLE_USAGE_LIST('SALES', 'INVENTORYUL', -2))
ORDER BY ROWS_READ DESC
FETCH FIRST 10 ROWS ONLY
```

7. If you want to view the text of a statement that affected the table, use the value of the **executable_id** element in the MON_GET_TABLE_USAGE_LIST output as input for the MON_GET_PKG_CACHE_STMT table function. For example, issue the following command to view the text of a particular statement:

```
SELECT STMT_TEXT
FROM TABLE
(MON_GET_PKG_CACHE_STMT(NULL,
x'01000000000000007C0000000000000000000000020020081126171720728997',
NULL, -2))
```

8. Use the list of statements and the statistics that are provided for the statements to determine where additional monitoring or tuning, if any, is required. For example, a statement that has a low value for the **pool_writes** monitor element compared to the **direct_writes** monitor element value might have buffer pool issues that require attention.

What to do next

When you do not require the information in the usage list, free the memory that is associated with the usage list by using the SET USAGE LIST STATE statement. For example, to free the memory that is associated with the INVENTORYUL usage list, issue the following command:

```
SET USAGE LIST SALES.INVENTORYUL STATE = RELEASED
```

Identifying how a statement affects database objects

Use a section explain that includes section actuals information to identify how a statement affects database objects. You can use statistics about how the statement section affected each table or index to determine whether additional monitoring or tuning is required.

Before you begin

Do the following tasks:

- Identify a statement for which you want to view object usage statistics.
- Ensure that you migrated your explain tables to DB2 Version 10.1.
- Ensure that automatic statistics profile generation is not enabled.
- Ensure that you have the privileges that are required to call the EXPLAIN_FROM_ACTIVITY procedure.

About this task

After you identify a statement for which you want to view object usage statistics, you can get a section explain that includes section actuals information. Section actuals information indicates how the statement affected each table or index that the statement used when it executed.

Actuals information includes runtime statistics about factors like locks and buffer pool usage for each table or index. You can compare these statistics to baseline data and use them to determine where additional monitoring or tuning might be required.

Procedure

To determine how database objects are affected by a statement:

1. Enable the collection of section actuals at the database level by issuing the following command:
   ```
   DB2 UPDATE DATABASE CONFIGURATION USING SECTION_ACTUALS BASE
   ```
2. Create a workload to collect section actuals information for activities that are submitted by the application that issues the statement. For example, to create the ACTWORKLOAD workload for activities that are submitted by the TEST application and enable collection for those activities, issue the following command:
   ```
   CREATE WORKLOAD ACTWORKLOAD APPLNAME ('TEST')
   COLLECT ACTIVITY DATA ON ALL WITH DETAILS,SECTION INCLUDE ACTUALS BASE
   ```
   Enabling collection of section actuals can also be accomplished in the following ways:
   - The CREATE SERVICE CLASS or ALTER SERVICE CLASS statement
   - The CREATE WORK ACTION SET or ALTER WORK ACTION SET statement

   - The WLM_SET_CONN_ENV procedure
   - The **section_actuals** configuration parameter
3. Create an activity event monitor by using the CREATE EVENT MONITOR statement. For example, to create the ACTEVMON activity event monitor, issue the following command:

   ```
   CREATE EVENT MONITOR ACTEVMON
          FOR ACTIVITIES
          WRITE TO TABLE
          CONTROL (TABLE CONTROL_ACTEVMON ),
          ACTIVITY (TABLE ACTIVITY_ACTEVMON ),
          ACTIVITYSTMT (TABLE ACTIVITYSTMT_ACTEVMON ),
          ACTIVITYVALS (TABLE ACTIVITYVALS_ACTEVMON ),
          ACTIVITYMETRICS (TABLE ACTIVITYMETRICS_ACTEVMON )
   ```
4. Activate the activity event monitor that you created by using the SET EVENT MONITOR STATE statement. For example, to activate the ACTEVMON activity event monitor, issue the following command:

   ```
   SET EVENT MONITOR ACTEVMON STATE 1
   ```
5. Run the application that issues the statement for which you want to view object statistics.
6. Find identifier information for the statement section by using the following command to query the activity event monitor tables:

   ```
   SELECT APPL_ID,
          UOW_ID,
          ACTIVITY_ID,
          STMT_TEXT
   FROM ACTIVITYSTMT_ACTEVMON
   ```
7. Obtain a section explain with actuals by using the activity identifier information as input for the EXPLAIN_FROM_ACTIVITY procedure. For example, to obtain a section explain for a section with an application ID of *N2.DB2INST1.0B5A12222841, a unit of work ID of 16, and an activity ID of 4, issue the following command:

   ```
   CALL EXPLAIN_FROM_ACTIVITY( '*N2.DB2INST1.0B5A12222841', 16, 4, 'ACTEVMON',
    'MYSCHEMA', ?, ?, ?, ?, ? )
   ```

   You get output that looks like the following sample output:

   ```
   Value of output parameters
   --------------------------
   Parameter Name : EXPLAIN_SCHEMA
   Parameter Value : MYSCHEMA

   Parameter Name : EXPLAIN_REQUESTER
   Parameter Value : GSDBUSER3

   Parameter Name : EXPLAIN_TIME
   Parameter Value : 2010-11-23-10.51.09.631945

   Parameter Name : SOURCE_NAME
   Parameter Value : SQLC2J21

   Parameter Name : SOURCE_SCHEMA
   Parameter Value : NULLID

   Parameter Name : SOURCE_VERSION
   Parameter Value :

   Return Status = 0
   ```
8. Format the explain data by using the **db2exfmt** command. Use the values of the **explain_requester**, **explain_time**, **source_name**, **source_schema**, and

source_version parameters in the output from the EXPLAIN_FROM_ACTIVITY procedure as input for the command.

9. View the explain output to determine how the section affected the database objects that it used when it executed. Statistics in the output might indicate that additional monitoring or tuning is required. For example, if a table that the section uses has a high value for the **lock_wait** monitor element, lock management might be required.
10. If you tune the statement, repeat steps 5 on page 418 through 9 to verify that performance is improved.

What to do next

Deactivate the activity event monitor by using the SET EVENT MONITOR STATE statement. For example, to deactivate the ACTEVMON activity event monitor, issue the following command:

```
SET EVENT MONITOR ACTEVMON STATE 0
```

Monitoring locking using table functions

You can retrieve information about locks using table functions. Unlike request, activity or data object monitor elements, information about locks is always available from the database manager. You do not need to enable the collection of this information.

Use the following monitor table functions to access current information for locks in the system:

- MON_GET_LOCKS
- MON_GET_APPL_LOCKWAIT

Both table functions return data in relational form.

Monitoring system memory using table functions

You can retrieve information about system memory usage using table functions.

You can examine memory usage at the level of memory sets, which are allocations of memory from the operating system. You can also examine memory usage by specific memory pools within a given memory set. Use the following monitor functions to access current information about memory usage:

- MON_GET_MEMORY_SET
- MON_GET_MEMORY_POOL

Other monitoring table functions

Besides table functions that return information about the system, activities, locks, or data objects there are also table functions that return various types of miscellaneous information. These functions include ones that return information related to the fast communications manager (FCM), and about the status of table space extent movement.

Each of the table functions that follow can be used at any time. Unlike the table functions that return request metrics (the system monitoring perspective), activity metrics (the activity monitoring perspective) or metrics related to data objects (the data object monitoring perspective), it is not necessary to first enable the collection of the monitor elements returned by these functions.

- MON_GET_FCM

- MON_GET_FCM_CONNECTION_LIST
- MON_GET_EXTENT_MOVEMENT_STATUS

Interfaces that return monitor data in XML documents

Some monitor data is reported as elements in XML documents.

Using XML to report monitor information provides improved extensibility and flexibility. New monitor elements can be added without having to add new columns to an output table. Also, XML documents can be processed in a number of ways, depending on your needs. For example:

- You can use XQuery to run queries against the XML document.
- You can use the XSLTRANSFORM scalar function to transform the document into other formats.
- You can view their contents as formatted text by using built-in MON_FORMAT_XML_* formatting functions, or the XMLTABLE table function.

XML documents that contain monitor elements are produced by several monitoring interfaces. The sections that follow describe how results are returned as XML documents.

- “Monitor table functions with names that end with "_DETAILS"”
- “XML data returned by event monitors” on page 421.

Monitor table functions with names that end with “_DETAILS”

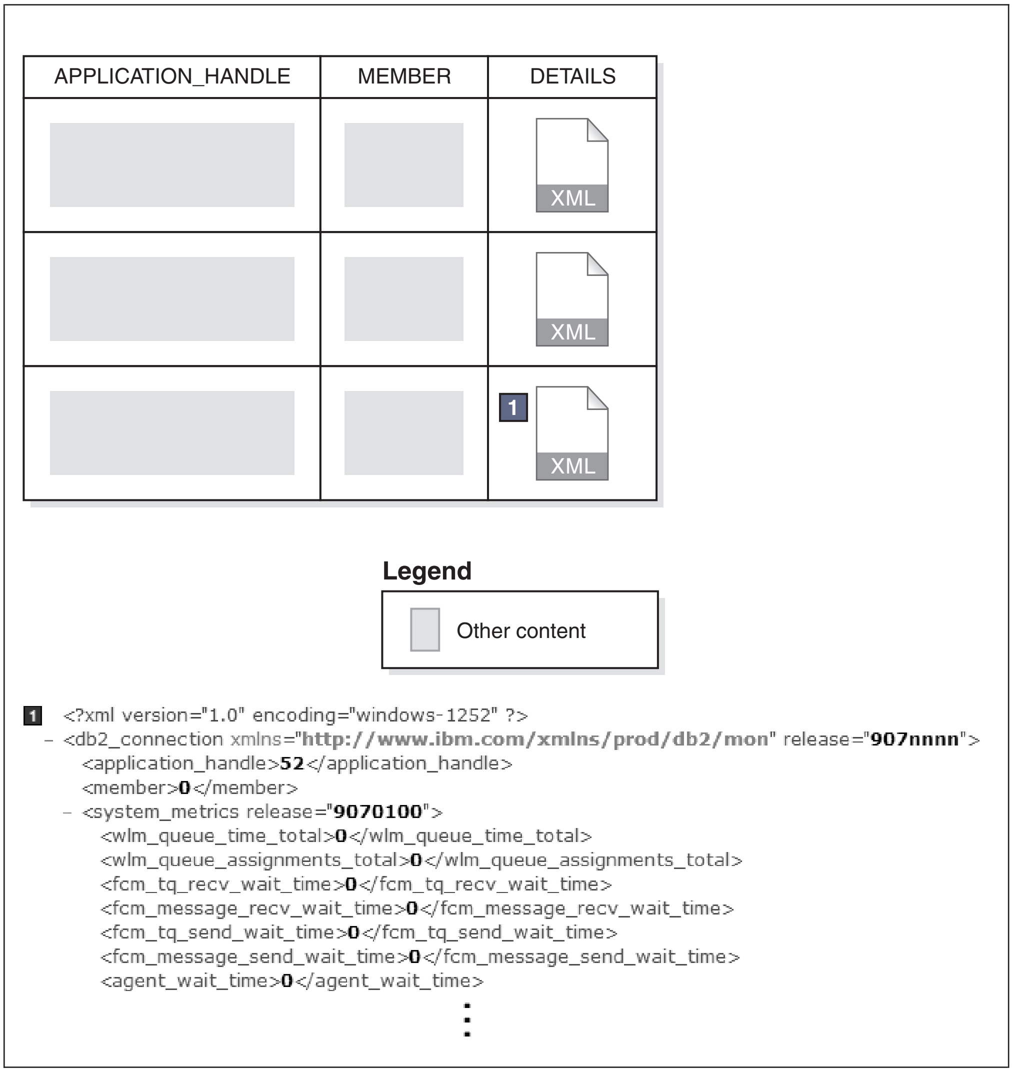
Examples of these table functions include:

- MON_GET_PKG_CACHE_STMT_DETAILS
- MON_GET_WORKLOAD_DETAILS
- MON_GET_CONNECTION_DETAILS
- MON_GET_SERVICE_SUBCLASS_DETAILS
- MON_GET_ACTIVITY_DETAILS
- MON_GET_UNIT_OF_WORK_DETAILS

These table functions return monitor elements from the system and the activity monitoring perspectives. Most of the monitor elements returned by these functions are contained in an XML document. For example, the MON_GET_CONNECTION_DETAILS table function returns the following columns:

- APPLICATION_HANDLE
- MEMBER
- DETAILS

The DETAILS column of each row contains an XML document that contains monitor element data. This XML document is composed of several document elements that correspond to monitor elements. Figure 37 on page 421 illustrates the DETAILS column that contains the XML documents. In addition, it shows monitor elements returned in the XML documents in the DETAILS column.



```
<?xml version="1.0" encoding="windows-1252" ?>
- <db2_connection xmlns="http://www.ibm.com/xmlns/prod/db2/mon" release="907nnnn">
    <application_handle>52</application_handle>
    <member>0</member>
  - <system_metrics release="9070100">
      <wlm_queue_time_total>0</wlm_queue_time_total>
      <wlm_queue_assignments_total>0</wlm_queue_assignments_total>
      <fcm_tq_recv_wait_time>0</fcm_tq_recv_wait_time>
      <fcm_message_recv_wait_time>0</fcm_message_recv_wait_time>
      <fcm_tq_send_wait_time>0</fcm_tq_send_wait_time>
      <fcm_message_send_wait_time>0</fcm_message_send_wait_time>
      <agent_wait_time>0</agent_wait_time>
                              ⋮
```

Figure 37. Table returned by MON_GET_CONNECTION_DETAILS, showing the DETAILS column that contains XML documents. The contents of the XML document in the third row (1) are shown following the table.

In the preceding example, the <agent_wait_time> XML document element corresponds to **agent_wait_time** monitor element.

The schema for the XML document that is returned in the DETAILS column is available in the file sqllib/misc/DB2MonRoutines.xsd. Further details can be found in the file sqllib/misc/DB2MonCommon.xsd.

Some of the monitor elements contained in the document in the DETAILS column might be grouped into higher-level document elements. For example, monitor elements that report on activity-related metrics are part of the **activity_metrics** element. Similarly, system-level metrics are part of the **system_metrics** element.

XML data returned by event monitors

Several event monitors return data in XML format. They are summarized in Table 76 on page 422. Details about the XML documents returned by the various event monitor are described in the sections that follow.

Table 76. XML documents returned by various event monitors

Event monitor	Event monitor output format	XML document returned
"Statistics event monitor"	Relational table File Named pipe	**metrics** The metrics reported in this document reflect the change in value for each metric since the last time statistics were collected. **details_xml** The metrics reported in this document accumulate until the database is deactivated.
"Activity event monitor" on page 423	Relational table File Named pipe	details_xml
"Package cache event monitor" on page 424	Unformatted event (UE) table	metrics This document can be viewed only after the UE table is transformed to either XML or relational tables.
"Unit of work event monitor" on page 424	Unformatted event (UE) table	metrics This document can be viewed only after the UE table is transformed to either XML or relational tables.

Statistics event monitor

The statistics event monitor records metrics in XML format when either of the two following logical data groups are included in the event monitor output:

- EVENT_SCSTATS
- EVENT_WLSTATS

When you create a statistics event monitor to report on monitor elements in either of these groups, some system metrics are collected as part of two XML documents. One for each of the **`details_xml`** and **`metrics`** monitor elements. Both XML documents contain the same set of monitor elements. In the `metrics` document, the values of the elements reflect the change in value for each element since the last time statistics were collected. The values of the elements contained in `details_xml` are not reset at each interval; they are reset only when the database is reactivated. If the data is written to a file or named pipe, these elements are part of the self-describing data stream. If the event monitor data is written to a table, the `metrics` document is stored in a column called METRICS; `details_xml` is stored in a column called DETAILS_XML. Figure 38 on page 423 shows the XML documents in the METRICS and DETAILS_XML columns as they appear in the SCSTATS table produced by the statistics event monitor:

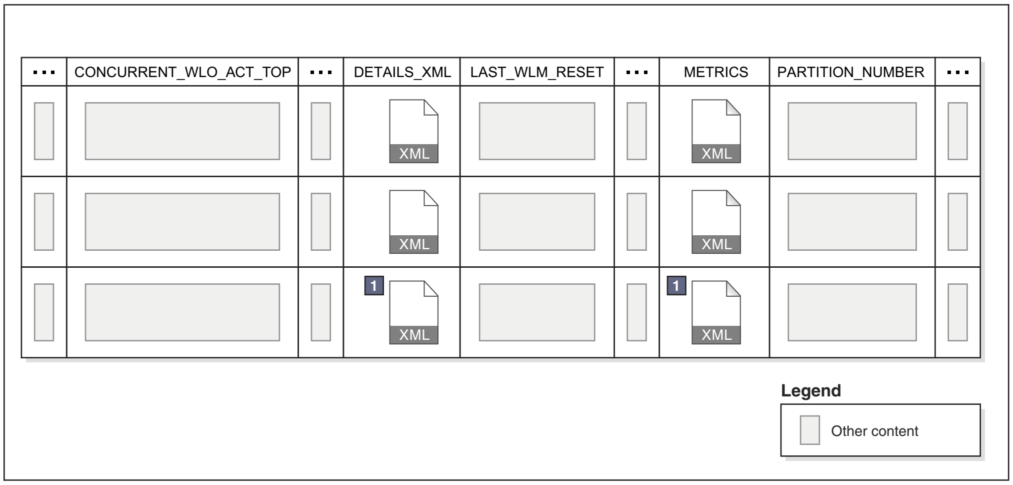
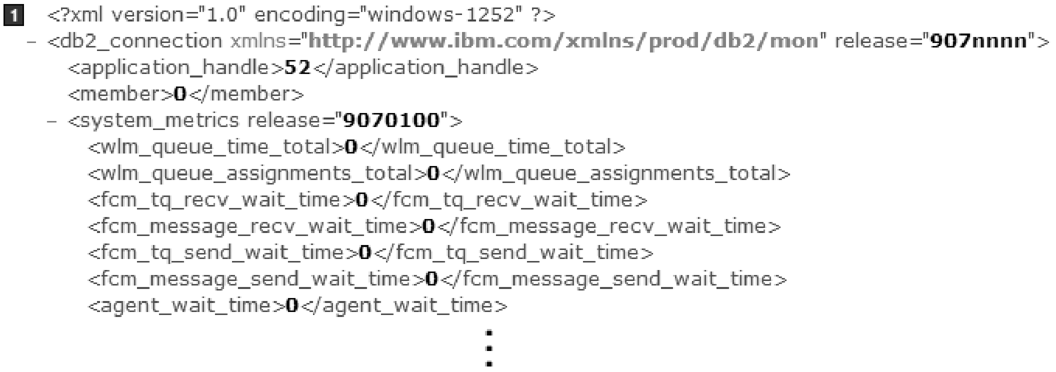


Figure 38. Output of statistics event monitor (when written to a table), showing the DETAILS_XML and METRICS columns.. The contents of the XML document in the third row (**1**) are shown following the table.

Each of the documents contained in these columns contains **system_metrics** as the top-level element, which, in turn, contains a number of monitor elements that report on system-related metrics.

1



```
  <?xml version="1.0" encoding="windows-1252" ?>
- <db2_connection xmlns="http://www.ibm.com/xmlns/prod/db2/mon" release="907nnnn">
    <application_handle>52</application_handle>
    <member>0</member>
  - <system_metrics release="9070100">
      <wlm_queue_time_total>0</wlm_queue_time_total>
      <wlm_queue_assignments_total>0</wlm_queue_assignments_total>
      <fcm_tq_recv_wait_time>0</fcm_tq_recv_wait_time>
      <fcm_message_recv_wait_time>0</fcm_message_recv_wait_time>
      <fcm_tq_send_wait_time>0</fcm_tq_send_wait_time>
      <fcm_message_send_wait_time>0</fcm_message_send_wait_time>
      <agent_wait_time>0</agent_wait_time>
                              ⋮
```

In addition to viewing system metrics from the XML document in the **metrics** monitor element, you can view the individual metrics directly from the output associated with the EVENT_SCMETRICS and EVENT_WLMETRICS logical data groups.

Notes:

- The **system_metrics** element that is reported in the details_xml document contained in the DETAILS_XML column produced by the statistics event monitor is also a part of the XML document contained in the DETAILS column returned by the MON_GET_SERVICE_SUBCLASS_DETAILS and MON_GET_WORKLOAD_DETAILS table functions. Like the metrics reported in the details_xml document, the values for the metrics reported in the document contained in the DETAILS column accumulate until the database is deactivated.

Activity event monitor

When you create an activity event monitor to report on monitor elements in the event_activity logical data group, one of the columns produced is DETAILS_XML. If the event monitor is written to a table, DETAILS_XML is a column. If it is written to a file or named pipe, DETAILS_XML is part of the self-describing data stream. Either way, the document contains the **activity_metrics** monitor element, which, in turn, contains a number of monitor elements that report on metrics related to activities.

Note: activity_metrics as reported in the XML document in the DETAILS_XML column produced by the activity event monitor is also a part of the XML document contained in the DETAILS column returned by the MON_GET_ACTIVITY_DETAILS table function.

Package cache event monitor

The package cache event monitor writes its output to an unformatted event (UE) table. If you convert the data in this table with the EVMON_FORMAT_UE_TO_TABLES table function, one of the tables produced is PKGCACHE_EVENT. This table contains a METRICS column. In each row, this column contains an XML document with elements associated with package cache event monitor elements.

Note: The EVMON_FORMAT_UE_TO_TABLES function also creates a separate table for the metrics collected by this event monitor called PKGCACHE_METRICS. This table contains the same information reported in the METRICS column of the PKGCACHE_EVENT table. So, you can retrieve metrics from the columns of the PKGCACHE_METRICS table, or you can use the XML document contained in the METRICS column of the PKGCACHE_EVENT table.

The EVMON_FORMAT_UE_TO_XML table function also produces an XML document with elements associated with package cache event monitor elements. For example, the XML document element <num_executions> corresponds to the **num_executions** monitor element.

Unit of work event monitor

The unit of work event monitor writes its output to an unformatted event (UE) table. If you convert the data in this table with the EVMON_FORMAT_UE_TO_TABLES table function, one of the tables produced is UOW_EVENT. This table contains a METRICS column, which contains an XML document with elements associated with unit of work event monitor elements.

The EVMON_FORMAT_UE_TO_XML function also produces an XML document with elements associated with unit of work event monitor elements. For example, the XML document element <workload_name> corresponds to the **workload_name** monitor element.

Snapshot monitor

You can use the snapshot monitor to capture information about the database and any connected applications at a specific time. Snapshots are useful for determining the status of a database system.

Taken at regular intervals, they are also useful for observing trends and foreseeing potential problems. Some of the data from the snapshot monitor is obtained from the system monitor. The data available from the system monitor is determined by system monitor switches.

The system monitor accumulates information for a database only while it is active. If all applications disconnect from a database and the database deactivates, then the system monitor data for that database is no longer available. You can keep the database active until your final snapshot has been taken, either by starting the database with the ACTIVATE DATABASE command, or by maintaining a permanent connection to the database.

Snapshot monitoring requires an instance attachment. If there is not an attachment to an instance, then a default instance attachment is created. An instance attachment is usually done implicitly to the instance specified by the DB2INSTANCE environment variable when the first database system monitor API is invoked by the application. It can also be done explicitly, using the ATTACH TO command. Once an application is attached, all system monitor requests that it invokes are directed to that instance. This allows a client to monitor a remote server by simply attaching to the instance on it.

In partitioned database environments, snapshots can be taken at any partition of the instance, or globally using a single instance connection. A global snapshot aggregates the data collected at each partition and returns a single set of values.

In DB2 pureScale environments, snapshots can be taken at any member or globally. A global snapshot aggregates the data collected at each member and returns a single set of values.

You can capture a snapshot from the CLP, from SQL table functions, or by using the snapshot monitor APIs in a C or C++ application. A number of different snapshot request types are available, each returning a specific type of monitoring data. For example, you can capture a snapshot that returns only buffer pool information, or a snapshot that returns database manager information. Before capturing a snapshot, consider if you need information from monitor elements that are under monitor switch control. If a particular monitor switch is off, the monitor elements under its control will not be collected.

Access to system monitor data: SYSMON authority

Users that are part of the SYSMON database manager level group have the authority to gain access to database system monitor data. System monitor data is accessed using the snapshot monitor APIs, CLP commands, or SQL table functions.

The SYSMON authority group provides the means to enable users without system administration or system control authorities to access database system monitor data.

Aside from SYSMON authority, the only way to access system monitor data using the snapshot monitor is with system administration or system control authority.

Any user that is part of the SYSMON group or has system administration or system control authority can perform the following snapshot monitor functions:

- CLP Commands:
  - GET DATABASE MANAGER MONITOR SWITCHES
  - GET MONITOR SWITCHES
  - GET SNAPSHOT
  - LIST ACTIVE DATABASES
  - LIST APPLICATIONS
  - LIST DCS APPLICATIONS

  - LIST UTILITIES
  - RESET MONITOR
  - UPDATE MONITOR SWITCHES
- APIs:
  - db2GetSnapshot - Get Snapshot
  - db2GetSnapshotSize - Estimate Size Required for db2GetSnapshot() Output Buffer
  - db2MonitorSwitches - Get/Update Monitor Switches
  - db2ResetMonitor - Reset Monitor
- Snapshot SQL table functions without previously running SYSPROC.SNAP_WRITE_FILE

Capturing database system snapshots by using snapshot administrative views and table functions

Authorized users can capture snapshots of monitor information for a DB2 instance by using snapshot administrative views or snapshot table functions. The snapshot administrative views provide a simple means of accessing data for all database partitions of the connected database.

The snapshot table functions allow you to request data for a specific database partition, globally aggregated data, or data from all database partitions. Some snapshot table functions allow you to request data from all active databases.

Before you begin

You must have SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority to capture a database snapshot. To obtain a snapshot of a remote instance, you must first connect to a local database belonging to that instance.

About this task

While new snapshot table functions might be required in future releases if new monitor data is available, the set of snapshot administrative views will remain the same with new columns added to the view, making the administrative views a good choice for application maintenance over time.

Each snapshot view returns a table with one row per monitored object per database partition with each column representing a monitor element. Each table function returns a table with one row per monitored object for the specified partition. The column names of the returned table correlate with the monitor element names.

For example, a snapshot of general application information for the SAMPLE database is captured as follows by using the SNAPAPPL administrative view:

```
SELECT * FROM SYSIBMADM.SNAPAPPL
```

You can also select individual monitor elements from the returned table. For example, the following statement returns only the **agent_id** and **db_name** monitor elements:

```
SELECT agent_id, db_name FROM SYSIBMADM.SNAPAPPL
```

Restrictions

Snapshot administrative views and table functions cannot be used with either of the following:

- Monitor switches commands or APIs
- Monitor reset commands or APIs

This restriction includes:

- **GET MONITOR SWITCHES**
- **UPDATE MONITOR SWITCHES**
- **RESET MONITOR**

This limitation is because such commands use an ATTACH command, while snapshot table functions use CONNECT statement..

Procedure

- To capture a snapshot using a snapshot administrative view:
  1. Connect to a database. This can be any database in the instance you need to monitor. To be able to issue an SQL query with a snapshot administrative view, you must be connected to a database.
  2. Determine the type of snapshot you need to capture. If you want to capture a snapshot for a database other than the currently connected database, or if you want to retrieve data from a single database partition, or global aggregate data, you need to use a snapshot table function instead.
  3. Issue a query with the appropriate snapshot administrative view. For example, here is a query that captures a snapshot of lock information for the currently connected database:

     ```
     SELECT * FROM SYSIBMADM.SNAPLOCK
     ```
- To capture a snapshot using a snapshot table function:
  1. Connect to a database. This can be any database in the instance you need to monitor. To be able to issue an SQL query with a snapshot table function, you must be connected to a database.
  2. Determine the type of snapshot you need to capture.
  3. Issue a query with the appropriate snapshot table function. For example, here is a query that captures a snapshot of lock information about the SAMPLE database for the current connected database partition:

     ```
     SELECT * FROM TABLE(SNAP_GET_LOCK('SAMPLE',-1)) AS SNAPLOCK
     ```

     The SQL table functions have two input parameters:

     database name
        VARCHAR(255). If you enter NULL, the name of the currently connected database is used.

     partition number
        SMALLINT. For the database partition number parameter, enter the integer (a value between 0 and 999) corresponding to the database partition number you need to monitor. To capture a snapshot for the currently connected database partition, enter a value of -1. To capture a global aggregate snapshot, enter a value of -2. To capture a snapshot from all database partitions, do not specify a value for this parameter.

     Note:

     a. For the following list of snapshot table functions, if you enter a NULL for the currently connected database, you will get snapshot information for all databases in the instance:

- SNAP_GET_DB
- SNAP_GET_DB_MEMORY_POOL
- SNAP_GET_DETAILLOG
- SNAP_GET_HADR
- SNAP_GET_STORAGE_PATHS
- SNAP_GET_APPL
- SNAP_GET_APPL_INFO
- SNAP_GET_AGENT
- SNAP_GET_AGENT_MEMORY_POOL
- SNAP_GET_STMT
- SNAP_GET_SUBSECTION
- SNAP_GET_BP
- SNAP_GET_BP_PART

b. The database name parameter does not apply to the database manager level snapshot table functions; they have only a parameter for database partition number. The database partition number parameter is optional.

Capturing database system snapshot information to a file using the SNAP_WRITE_FILE stored procedure

With the SNAP_WRITE_FILE stored procedure you can capture snapshots of monitor data and save this information to files on the database server and allow access to the data by users who do not have SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority.

Any user can then issue a query with a snapshot table function to access the snapshot information in these files. In providing open access to snapshot monitor data, sensitive information (such as the list of connected users and the SQL statements they have submitted to the database) is available to all users who have the execution privilege for the snapshot table functions. The privilege to execute the snapshot table functions is granted to PUBLIC by default. (Note, however, that no actual data from tables or user passwords can be exposed using the snapshot monitor table functions.)

Before you begin

You must have SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority to capture a database snapshot with the SNAP_WRITE_FILE stored procedure.

About this task

When issuing a call to the SNAP_WRITE_FILE stored procedure, in addition to identifying the database and partition to be monitored, you need to specify a *snapshot request type*. Each snapshot request type determines the scope of monitor data that is collected. Choose the snapshot request types based on the snapshot table functions users will need to run. The following table lists the snapshot table functions and their corresponding request types.

Table 77. Snapshot request types

Snapshot table function	Snapshot request type
SNAP_GET_AGENT	APPL_ALL
SNAP_GET_AGENT_MEMORY_POOL	APPL_ALL

Table 77. Snapshot request types (continued)

Snapshot table function	Snapshot request type
SNAP_GET_APPL	APPL_ALL
SNAP_GET_APPL_INFO	APPL_ALL
SNAP_GET_STMT	APPL_ALL
SNAP_GET_SUBSECTION	APPL_ALL
SNAP_GET_BP_PART	BUFFERPOOLS_ALL
SNAP_GET_BP	BUFFERPOOLS_ALL
SNAP_GET_DB	DBASE_ALL
SNAP_GET_DETAILLOG	DBASE_ALL
SNAP_GET_DB_MEMORY_POOL	DBASE_ALL
SNAP_GET_HADR	DBASE_ALL
SNAP_GET_STORAGE_PATHS	DBASE_ALL
SNAP_GET_DBM	DB2
SNAP_GET_DBM_MEMORY_POOL	DB2
SNAP_GET_FCM	DB2
SNAP_GET_FCM_PART	DB2
SNAP_GET_SWITCHES	DB2
SNAP_GET_DYN_SQL	DYNAMIC_SQL
SNAP_GET_LOCK	DBASE_LOCKS
SNAP_GET_LOCKWAIT	APPL_ALL
SNAP_GET_TAB	DBASE_TABLES
SNAP_GET_TAB_REORG	DBASE_TABLES
SNAP_GET_TBSP	DBASE_TABLESPACES
SNAP_GET_TBSP_PART	DBASE_TABLESPACES
SNAP_GET_CONTAINER	DBASE_TABLESPACES
SNAP_GET_TBSP_QUIESCER	DBASE_TABLESPACES
SNAP_GET_TBSP_RANGE	DBASE_TABLESPACES
SNAP_GET_UTIL	DB2
SNAP_GET_UTIL_PROGRESS	DB2

Procedure

1. Connect to a database. This can be any database in the instance you need to monitor. To be able to call a stored procedure, you must be connected to a database.
2. Determine the snapshot request type, and the database and partition you need to monitor.
3. Call the SNAP_WRITE_FILE stored procedure with the appropriate parameter settings for the snapshot request type, database, and partition. For example, here is a call that will capture a snapshot of application information about the SAMPLE database for the current connected partition:

   ```
   CALL SNAP_WRITE_FILE('APPL_ALL','SAMPLE',-1)
   ```

   The SNAP_WRITE_FILE stored procedure has three input parameters:

- a snapshot request type (see Table 77 on page 428, which provides a cross-reference of the snapshot table functions and their corresponding request types)
- a VARCHAR (128) for the database name. If you enter NULL, the name of the currently connected database is used.

  Note: This parameter does not apply to the database manager level snapshot table functions; they only have parameters for request type and partition number.
- a SMALLINT for the partition number (a value between 0 and 999). For the partition number parameter, enter the integer corresponding to partition number you want to monitor. To capture a snapshot for the currently connected partition, enter a value of -1 or a NULL. To capture a global snapshot, enter a value of -2.

Results

Once the snapshot data has been saved to a file, all users can issue queries with the corresponding snapshot table functions, specifying (NULL, NULL) as input values for database-level table functions, and (NULL) for database manager level table functions. The monitor data they receive is pulled from the files generated by the SNAP_WRITE_FILE stored procedure.

Note: While this provides a means to limit user access to sensitive monitor data, this approach does have some limitations:

- The snapshot monitor data available from the SNAP_WRITE_FILE files is only as recent as the last time the SNAP_WRITE_FILE stored procedure was called. You can ensure that recent snapshot monitor data is available by making calls to the SNAP_WRITE_FILE stored procedure at regular intervals. For instance, on UNIX systems you can set a cron job to do this.
- Users issuing queries with the snapshot table functions cannot identify a database or partition to monitor. The database name and partition number identified by the user issuing the SNAP_WRITE_FILE calls determine the contents of the files accessible by the snapshot table functions.
- If a user issues an SQL query containing a snapshot table function for which a corresponding SNAP_WRITE_FILE request type has not been run, a direct snapshot is attempted for the currently connected database and partition. This operation is successful only if the user has SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority.

Accessing database system snapshots using snapshot table functions in SQL queries (with file access)

For every request type that authorized users have called the SNAP_WRITE_FILE stored procedure, any user can issue queries with the corresponding snapshot table functions. The monitor data they receive will be retrieved from the files generated by the SNAP_WRITE_FILE stored procedure.

Before you begin

For every snapshot table function with which you intend to access SNAP_WRITE_FILE files, an authorized user must have issued a SNAP_WRITE_FILE stored procedure call with the corresponding snapshot request types. If you issue an SQL query containing a snapshot table function for which a corresponding SNAP_WRITE_FILE request type has not been run, a direct snapshot is attempted for the currently connected database and partition. This operation is successful only if the user has SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority.

About this task

Users who access snapshot data from SNAP_WRITE_FILE files with snapshot table functions cannot identify a database or partition to monitor. The database name and partition number identified by the user issuing the SNAP_WRITE_FILE calls determine the contents of the SNAP_WRITE_FILE files. The snapshot monitor data available from the SNAP_WRITE_FILE files is only as recent as the last time the SNAP_WRITE_FILE stored procedure captured snapshots.

Procedure

1. Connect to a database. This can be any database in the instance you need to monitor. To issue an SQL query with a snapshot table function, you must be connected to a database.
2. Determine the type of snapshot you need to capture.
3. Issue a query with the appropriate snapshot table function. For example, here is a query that will capture a snapshot of table space information:

   ```
   SELECT * FROM TABLE(SNAP_GET_TBSP(CAST(NULL AS VARCHAR(1)),
                       CAST (NULL AS INTEGER))) AS SNAP_GET_TBSP
   ```

   Note: You must enter NULL values for the database name and partition number parameters. The database name and partition for the snapshot are determined in the call of the SNAP_WRITE_FILE stored procedure. Also, the database name parameter does not apply to the database manager level snapshot table functions; they only have a parameter for partition number. Each snapshot table function returns a table with one or more rows, with each column representing a monitor element. Accordingly, the monitor element column names correlate to the monitor element names.
4. You can also select individual monitor elements from the returned table. For example, the following statement will return only the **agent_id** monitor element:

   ```
   SELECT agent_id FROM TABLE(
                              SNAP_GET_APPL(CAST(NULL AS VARCHAR(1)),
                                            CAST(NULL AS INTEGER)))
                              as SNAP_GET_APPL
   ```

Snapshot monitor SQL Administrative Views

There are a number of different snapshot monitor SQL administrative views available, each returning monitor data about a specific area of the database system. For example, the SYSIBMADM.SNAPBP SQL administrative view captures a snapshot of buffer pool information.

The following table lists each available snapshot monitor administrative view.

Table 78. Snapshot Monitor SQL Administrative Views

Monitor level	SQL Administrative Views	Information returned
Database manager	SYSIBMADM.SNAPDBM	Database manager level information.
Database manager	SYSIBMADM.SNAPFCM	Database manager level information regarding the fast communication manager (FCM).

Table 78. Snapshot Monitor SQL Administrative Views (continued)

Monitor level	SQL Administrative Views	Information returned
Database manager	SYSIBMADM.SNAPFCM_PART	Database manager level information for a partition regarding the fast communication manager (FCM).
Database manager	SYSIBMADM.SNAPSWITCHES	Database manager monitor switch settings.
Database manager	SYSIBMADM.SNAPDBM_MEMORY_POOL	Database manager level information about memory usage.
Database	SYSIBMADM.SNAPDB	Database level information and counters for a database. Information is returned only if there is at least one application connected to the database.
Database	SYSIBMADM.SNAPDB_MEMORY_POOL	Database level information about memory usage for UNIX platforms only.
Application	SYSIBMADM.SNAPAPPL	General application level information for each application that is connected to the database. This includes cumulative counters, status information, and most recent SQL statement executed (if statement switch is set).
Application	SYSIBMADM.SNAPAPPL_INFO	General application level identification information for each application that is connected to the database.
Application	SYSIBMADM.SNAPLOCKWAIT	Application level information regarding lock waits for the applications connected to the database.
Application	SYSIBMADM.SNAPSTMT	Application level information regarding statements for the applications connected to the database. This includes the most recent SQL statement executed (if the statement switch is set).
Application	SYSIBMADM.SNAPAGENT	Application level information regarding the agents associated with applications connected to the database.
Application	SYSIBMADM.SNAPSUBSECTION	Application level information regarding the subsections of access plans for the applications connected to the database.
Application	SYSIBMADM.SNAPAGENT_MEMORY_POOL	Information about memory usage at the agent level.
Table	SYSIBMADM.SNAPTAB	Table activity information at the database and application level for each application connected to the database. Table activity information at the table level for each table that *was accessed* by an application connected to the database. Requires the table switch.
Table	SYSIBMADM.SNAPTAB_REORG	Table reorganization information at the table level for each table in the database undergoing reorganization.
Lock	SYSIBMADM.SNAPLOCK	Lock information at the database level, and application level for each application connected to the database. Requires the lock switch.

Table 78. Snapshot Monitor SQL Administrative Views (continued)

Monitor level	SQL Administrative Views	Information returned
Table space	SYSIBMADM.SNAPTBSP	Information about table space activity at the database level, the application level for each application connected to the database, and the table space level for each table space that has been accessed by an application connected to the database. Requires the buffer pool switch.
Table space	SYSIBMADM.SNAPTBSP_PART	Information about table space configuration.
Table space	SYSIBMADM.SNAPTBSP_QUIESCER	Information about quiescers at the table space level.
Table space	SYSIBMADM.SNAPCONTAINER	Information about table space container configuration at the table space level.
Table space	SYSIBMADM.SNAPTBSP_RANGE	Information about ranges for a table space map.
Buffer pool	SYSIBMADM.SNAPBP	Buffer pool activity counters for the specified database. Requires the buffer pool switch.
Buffer pool	SYSIBMADM.SNAPBP_PART	Information on buffer size and usage, calculated per partition.
Dynamic SQL	SYSIBMADM.SNAPDYN_SQL	Point-in-time statement information from the SQL statement cache for the database.
Database	SYSIBMADM.SNAPUTIL	Information about utilities.
Database	SYSIBMADM.SNAPUTIL_PROGRESS	Information about the progress of utilities.
Database	SYSIBMADM.SNAPDETAILLOG	Database level information about log files.
Database	SYSPROC.ADMIN_GET_STORAGE_PATHS	Returns a list of automatic storage paths for the database that includes file system information for each storage path.

Before capturing a snapshot, consider if you need information from monitor elements that are under monitor switch control. If a particular monitor switch is off, the monitor elements under its control will not be collected. See the individual monitor elements to determine if an element you need is under switch control.

All snapshot monitoring administrative views and associated table functions use a separate instance connection, which is different from the connection the current session uses. Therefore, an implicit instance attachment might be established, and only default database manager monitor switches are effective. Ineffective monitor switches include any that are turned on or off dynamically from the current session or application.

Also, there is a set of administrative views that do not only return values of individual monitor elements, but also return computed values that are commonly required in monitoring tasks. For example, the SYSIBMADM.BP_HITRATIO administrative view returns calculated values for buffer pool hit ratios, which combine a number of individual monitor elements.

Table 79. Snapshot Monitor SQL Administrative Convenience Views

SQL Administrative Convenience Views	Information returned
SYSIBMADM.APPLICATIONS	Information about connected database applications.

Table 79. Snapshot Monitor SQL Administrative Convenience Views (continued)

SQL Administrative Convenience Views	Information returned
SYSIBMADM.APPL_PERFORMANCE	Information about the rate of rows selected versus the number of rows read by an application.
SYSIBMADM.BP_HITRATIO	Buffer pool hit ratios, including total, data, and index, in the database.
SYSIBMADM.BP_READ_IO	Information about buffer pool read performance.
SYSIBMADM.BP_WRITE_IO	Information about buffer pool write performance.
SYSIBMADM.CONTAINER_UTILIZATION	Information about table space containers and utilization rates.
SYSIBMADM.LOCKS_HELD	Information on current locks held.
SYSIBMADM.LOCKWAITS	Information about DB2 agents working on behalf of applications that are waiting to obtain locks.
SYSIBMADM.LOG_UTILIZATION	Information about log utilization for the currently connected database.
SYSIBMADM.LONG_RUNNING_SQL	Information about the longest running SQL in the currently connected database.
SYSIBMADM.QUERY_PREP_COST	Information about the time required to prepare different SQL statements.
SYSIBMADM.TBSP_UTILIZATION	Table space configuration and utilization information.
SYSIBMADM.TOP_DYNAMIC_SQL	The top dynamic SQL statements sortable by number of executions, average execution time, number of sorts, or sorts per statement.

Event monitors

Monitoring table functions and snapshot routines return the values of monitor elements at the specific point in time the routine is run, which is useful when you want to check the current state of your system. However, you might not always want to monitor points in time.

There are many times when you need to capture information about the state of your system at exactly the time that a specific event occurs. Event monitors serve this purpose.

Event monitors can be created to capture point-in-time information related to different kinds of *events* that take place in your system. For example, you can create an event monitor to capture information when a specific threshold that you define is exceeded. The information captured includes such things as the ID of the application that was running when the threshold was exceeded. Or, you might create an event monitor to determine what statement was running when a lock event occurred.

Types of events for which event monitors capture data

You can use event monitors to capture information related to many different kinds of events that take place on your system.

The following tables lists the types of events that occur in the system that you can monitor with an event monitor. It also describes the type of data collected for different events, as well as when the monitoring data is collected. The names of the event monitors shown in column two correspond to the keywords used to create that type of event monitor using the CREATE EVENT MONITOR statement.

Table 80. Event Types

Type of event to monitor	Event monitor name	Event monitor properties	Details
Locks and deadlocks	LOCKING	*Uses of this event monitor*	To determine when locks or deadlocks occur, and the applications that are involved. The advantages of using the LOCKING event monitor instead of the deprecated DEADLOCKS event monitor include consolidated reporting of both lock and deadlock events, as well as the inclusion of information about lock waits and lock time-outs.
		Data collected	Comprehensive information regarding applications involved, including the identification of participating statements (and statement text) and a list of locks being held.
		When the event data is generated[1]	Upon detection of any of the following event types, depending on how you configure the event monitor: • lock timeout • deadlock • lock wait beyond a specified duration
Execution of a SQL statements or other operation that spawns a database activity.	ACTIVITIES	*Uses of this event monitor*	To track the execution of individual statements and other activities to understand what activities are running in the system. Also to capture activities for diagnostic reasons, and to study the resource consumption of SQL.
		Data collected	Activity level data, generally for activities involving workload management objects. • If WITH DETAILS was specified as part of COLLECT ACTIVITY DATA clause on the CREATE or ALTER statements for a workload management object, then information collected includes statement and compilation environment information for those activities that have it. If WITH SECTION is also specified, then statement, compilation environment, section environment data, and section actuals are also captured. • If AND VALUES was also specified on the CREATE OR ALTER statement for the workload management object, the information collected will also include input data values for those activities that have it.
		When event data is generated[1]	• Upon completion of an activity that executed in a service class, workload or work class that had its COLLECT ACTIVITY DATA option turned on. • When an activity violates a threshold that has the COLLECT ACTIVITY DATA option enabled. • At the instant the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure is executed. • When an activity is executed by a connection for which activity collection has been enabled using the WLM_SET_CONN_ENV stored procedure.

Table 80. Event Types (continued)

Type of event to monitor	Event monitor name	Event monitor properties	Details
Execution of an SQL statement	STATEMENTS	*Uses of this event monitor*	To see what requests are being made to the database as a result of the execution of SQL statements.
		Data collected	Statement start or stop time, CPU used, text of dynamic SQL, SQLCA (return code of SQL statement), and other metrics such as fetch count. For partitioned databases: CPU used, execution time, table and table queue information. **Notes:** • When monitoring the execution of SQL procedures using statement event monitors, data manipulation language (DML) statements, such as INSERT, SELECT, DELETE, and UPDATE, generate events. Procedural statements, such as variable assignments and control structures (for example, WHILE or IF), do not generate events in a deterministic fashion. • Statement start or stop time is unavailable when the Timestamp switch is off.
		When event data is generated	End of SQL statement[2]; for partitioned databases, End of subsection[2]
Completion of a unit of work (transaction)	UNIT OF WORK	*Uses of this event monitor*	To gather resource usage information and performance metrics for units of work that run on the system. This information can be used for purposes ranging from generating reports for billing or charge-back purposes of system resources used by an application, to troubleshooting performance problems caused by slow-running routines. Recommended over the TRANSACTIONS event monitor.
		Data collected	Information about units of work (transactions), such as start and stop time, the workload and service class under which they ran. Option to include information about packages or executable IDs for statements run as part of the unit of work, as well as request metrics.
		When event data is generated[1]	Upon completion of a unit of work
Eviction of sections from the package cache	PACKAGE CACHE	*Uses of this event monitor*	To capture a history of statements (and related metrics) that are no longer in the package cache. This information can be used if you need to examine performance metrics for statements that are no longer available in memory.
		Data collected	Includes statement text and metrics aggregated over all executions of the section.
		When event data is generated[1]	As entries are evicted from the package cache.
Connections to the database by applications	CONNECTIONS	*Uses of this event monitor*	To capture metrics and other monitor elements for each connection to the database by an application.
		Data collected	All application-level counters. For example, the time that the application connected to or disconnected from the database, or number of lock escalations that the application was involved with.
		When event data is generated	End of connection[2]

Table 80. Event Types (continued)

Type of event to monitor	Event monitor name	Event monitor properties	Details
Deactivation of database	DATABASE	*Uses of this event monitor*	To capture metrics and other monitor elements that reflect information about the database as whole, since activation.
		Data collected	All database level counters. For example, the number of connections made to a database, time spent waiting on locks, or rows of data inserted since its activation.
		When event data is generated	Database deactivation[2]
	BUFFERPOOLS TABLESPACES	*Uses of this event monitor*	To capture metrics related to buffer pools and table spaces.
		Data collected	Counters for buffer pools, prefetchers, page cleaners and direct I/O for each buffer pool.
		When event data is generated	Database deactivation[2]
	TABLES	*Uses of this event monitor*	To capture metrics related to tables that have changed since database activation.
		Data collected	Table level counters, such as rows read or written, or disk pages used by data,LOB or index objects.
		When event data is generated	Database deactivation[2]
Statistics and metrics on workload management objects	STATISTICS	*Uses of this event monitor*	To capture processing metrics related to workload management objects (for example service superclasses, or workloads) in the database. For example, you could use a statistics event monitor to check on CPU utilization over time for a given workload.
		Data collected	Statistics computed from the activities that executed within each service class, workload, or work class that exists on the system.
		When event data is generated	Statistics can be collected automatically at regular intervals. This interval is defined with the **`wlm_collect_int`** database configuration parameter. Data can also collected manually, using the WLM_COLLECT_STATS stored procedure. **Note:** With either collection mechanism, the values of statistics monitor elements are reset to 0 after collection has taken place.
Exceeding a workload manager threshold	THRESHOLD VIOLATIONS	*Uses of this event monitor*	To determine when specific thresholds that you set are exceeded during database operations. Thresholds can be set for a variety of things, ranging from CPU time to the number of database connections, to the execution of specific statements. Data collected can be used for a variety of purposes, including monitoring for potential problems (such as approaching limits on temporary table space).
		Data collected	Threshold violation information.
		When event data is generated	Upon detection of a threshold violation. Thresholds are defined using the CREATE THRESHOLD statement.

Table 80. Event Types (continued)

Type of event to monitor	Event monitor name	Event monitor properties	Details
Changes to database or database manager configuration	CHANGE HISTORY	*Uses of this event monitor*	To captures change to database and database manager configuration and registry settings, execution of DDL statements, and execution of utilities
		Data collected	Database and database manager configuration parameter changes, registry variable changes, execution of DDL statements, execution of certain DB2 utilities and commands, and change history event monitor startup. **Note:** Generally, information related to events that occur while the change history event monitor is inactive or the database is offline are not captured. However, changes to registry variables and configuration parameters are recorded.
		When event data is generated[1]	Upon monitor startup, when a parameter or variable changes, or when a command, DDL, or utility completes.

Notes:

1. If a database is deactivated while an activity event monitor is active, backlogged activity records in the queue are discarded. To ensure that you obtain all activities event monitor records and that none are discarded, deactivate the activities event monitor before deactivating the database. When an activities event monitor is explicitly deactivated, all backlogged activity records in the queue are processed before the event monitor deactivates.
2. In addition to the defined times where data collection automatically occurs, you can use the FLUSH EVENT MONITOR SQL statement to generate events. The events generated by this method are written with the current database monitor values for all the monitor types (except for DEADLOCKS and DEADLOCKS WITH DETAILS) associated with the flushed event monitor.

Table 81. Event Types For Deprecated Event Monitors

Type of event to monitor	Event monitor name	Event monitor properties	Details
Deadlocks	DEADLOCKS[2]	*Uses of this event monitor*	To determine when deadlocks occur, and the applications that are involved.
		Data collected	Applications involved, and locks in contention.
		When event data is generated	Detection of a deadlock
	DEADLOCKS WITH DETAILS[2]	*Uses of this event monitor*	To determine when deadlocks occur, and the applications that are involved.
		Data collected	Comprehensive information regarding applications involved, including the identification of participating statements (and statement text) and a list of locks being held. Using a DEADLOCKS WITH DETAILS event monitor instead of a DEADLOCKS event monitor will incur a performance cost when deadlocks occur, due to the extra information that is collected.
		When event data is generated	Detection of a deadlock
	DEADLOCKS WITH DETAILS HISTORY[2]	*Uses of this event monitor*	To determine when deadlocks occur, and the applications that are involved.
		Data collected	All information reported in a DEADLOCKS WITH DETAILS event monitor, along with the statement history for the current unit of work of each application owning a lock participating in a deadlock scenario for the database partition where that lock is held. Using a DEADLOCKS WITH DETAILS HISTORY event monitor will incur a minor performance cost when activated due to statement history tracking.
		When event data is generated	Detection of a deadlock
	DEADLOCKS WITH DETAILS HISTORY VALUES[2]	*Uses of this event monitor*	
		Data collected	All information reported in a deadlock with details and history, along with the values provided for any parameter markers at the time of execution of a statement. Using a DEADLOCKS WITH DETAILS HISTORY VALUES event monitor will incur a more significant performance cost when activated due to extra copying of data values.
		When event data is generated	Detection of a deadlock
Completion of a unit of work (transaction)	TRANSACTIONS[3]	*Uses of this event monitor*	
		Data collected	UOW work start or stop time, previous UOW time, CPU consumed, locking and logging metrics. Transaction records are not generated if running with XA.
		When event data is generated	Upon completion of a unit of work[1]

Notes:

1. In addition to the defined times where data collection automatically occurs, you can use the FLUSH EVENT MONITOR SQL statement to generate events. The events generated by this method are written with the current database monitor values for all the monitor types (except for DEADLOCKS and DEADLOCKS WITH DETAILS) associated with the flushed event monitor.
2. This event monitor has been deprecated. Its use is no longer recommended and might be removed in a future release. Use the CREATE EVENT MONITOR FOR LOCKING statement to monitor lock-related events, such as lock timeouts, lock waits, and deadlocks.
3. This event monitor has been deprecated. Its use is no longer recommended and might be removed in a future release. Use the CREATE EVENT MONITOR FOR UNIT OF WORK statement to monitor transaction events.

Note: A detailed deadlock event monitor is created for each newly created database. This event monitor, named DB2DETAILDEADLOCK, starts when the database is activated and will write to files in the database directory. You can avoid the additional processor time this event monitor requires by dropping it. The DB2DETAILDEADLOCK event monitor is deprecated. Its use is no longer recommended and might be removed in a future release. Use the CREATE EVENT MONITOR FOR LOCKING statement to monitor lock-related events, such as lock timeouts, lock waits, and deadlocks.

Event monitors that write to tables

Starting in DB2 V10.1, all event monitors can write output to regular tables that can be queried directly using SQL.

In addition, starting with DB2 V10.1, you can use the procedure EVMON_UPGRADE_TABLES to upgrade the tables produced by event monitors in earlier releases. This capability lets you more easily retain event monitor data as you upgrade your DB2 product.

Working with event monitors

Generally, the process of creating and using event monitors to capture information about the system when certain events occur is similar for all event monitor types. First you create the event monitor, then you enable data collection, and finally, you access the data gathered.

About this task

This topic provides an outline of the general steps to follow when working with event monitors.

Procedure

To use an event monitor to capture event information:

1. Create the event monitor. To create an event monitor, use the appropriate version of the CREATE EVENT MONITOR statement. When you create an event monitor, you must choose how to record the data the event monitor collects. All event monitors can write their output to relational tables; however, depending on your specific purposes, there are different options that might be more appropriate.
2. Activate the event monitor. To activate the event monitor, use the SET EVENT MONITOR STATE statement. For example, for an event monitor called **capturestats**, use the following command:

   ```
   SET EVENT MONITOR capturestats STATE 1
   ```

   To turn off data collection by the event monitor, use the following statement:

   ```
   SET EVENT MONITOR capturestats STATE 0
   ```

   By default, some event monitors activate automatically upon database activation; others require that you activate them manually. However, an event monitor created with the AUTOSTART option will not automatically be activated until the next database activation. Use the SET EVENT MONITOR STATE statement to force a recently-created event monitor into the active state. To determine whether an event monitor starts automatically, refer to the reference information for the relevant CREATE EVENT MONITOR statement.

3. Enable the collection of data. (Only for LOCKING, ACTIVITIES, STATISTICS, UNIT OF WORK and PACKAGE CACHE event monitors) Enabling data collection involves configuring the database manager to gather specific types of data to be recorded by event monitors.

   Not all event monitors require data collection to be enabled; for those that do not, such as the TABLE event monitor, creating and activating them is sufficient to cause data to be collected. The threshold violations event monitor also starts data collection automatically; however, in this case, you must also define the thresholds for which you want data captured using the CREATE THRESHOLD statement.

   For those event monitors that require data collection to be enabled, there are different options available to you. Depending on the type of event monitor you are working with, you might set a database configuration parameter to enable data collection across the entire database. Alternatively, you might choose to enable the collection of specific kinds of data for specific types of workload objects. For example, to configure the collection of basic information for a unit of work event monitor when any unit of work in the system finishes, you can set the **mon_uow_data** parameter to BASE. Alternatively, to capture unit of work information only for a specific workload, you can specify the COLLECT UNIT OF WORK DATA BASE clause as part of the CREATE WORKLOAD or ALTER WORKLOAD statements.
4. Run your applications or queries. After the event monitor has been created, and activated, and you have enabled data collection, run the applications or queries for which you want to collect data.
5. Optional: Deactivate the event monitor. After you run the applications or queries for which you want data collected, you can deactivate the event monitor using the SET EVENT MONITOR STATE statement. (see step 2 on page 440). Deactivating the event monitor is not necessary before proceeding to the next step, however leaving the event monitor active will result in disk space being used for data that you might not be interested in looking at.
6. Examine the data collected by the event monitor. Depending on the type of output the event monitor creates, there are different options for accessing the data collected. If the data is written directly to a relational table, you can use SQL to access the data contained in the table columns. On the other hand, if the event monitor writes to an unformatted event (UE) table, you must post-process the UE table using a command like **db2evmonfmt** or a procedure like EVMON_FORMAT_UE_TO_TABLES before you can view the event data.
7. Optional: Prune data that is no longer needed from the event monitor tables. For event monitors that you use on a regular basis, you might want to prune unneeded data from the tables. For example, if you use a unit of work event monitor to generate daily accounting reports about the system resources used by different applications, you might want to delete the current day's data from the event monitor tables once the reports have been generated.

   Tip: If you need to prune event monitor output regularly, consider using an unformatted event (UE) table to record event monitor output. Starting in DB2 V10.1, UE tables can be pruned automatically after data is transferred to regular tables.

Output options for event monitors

Event monitors can report the data they collect in a number of ways. All event monitors can write the data they collect to tables; some write to unformatted event (UE) tables, which can help improve performance. Others can also write directly to a file or named pipe.

Depending on how you want to use the information collected by event monitors, and on the type of event monitor, you can choose to have the output that the event monitors collect produced in different ways. The output types available include:

Regular tables

As of DB2 V10.1, all event monitors can write to regular tables that can be queried directly using SQL. For a given event, each of the monitor elements or metrics collected for the event is written to its own column in the table. This makes it possible to use a SELECT statement query the output to examine the values for a specific monitor element.

To create an event monitor that writes to tables, specify the WRITE TO TABLE clause in the CREATE EVENT MONITOR statement. Depending on the event monitor, one or more tables are created to contain the ouput, each table containing monitor elements that belong to a single logical group.

Tables can be stored in a table space of your choosing; however the target table of a CREATE EVENT MONITOR statement must be a non-partitioned table.

Note: There are two types of event monitors that write to tables. The first type includes event monitors created in Version 9.7 and later releases. These include the unit of work, package cache, locking and change history event monitor. As of DB2 Version 10.1, the first three of these event monitors can write their output to regular tables as an alternative to UE tables. The change history event monitor writes only to regular tables.

The second type are the event monitors implemented before DB2 Version 9.7. These include all other event monitors.

Generally, after an event monitor of either type has been created, they work in much the same way. That is, you can use SQL to directly access the data in the tables that they produce. However, the older event monitors in the second category have additional options that you can specify when creating the event monitor. In addition, only event monitors in the second category are capable of writing also to files and named pipes.

Unformatted event (UE) tables

UE tables were introduced in DB2 Version 9.7 for the new event monitors added in that release. UE tables are relational tables, however, they have only a limited number of columns. Most of the data associated with each event is written to a column containing an inline binary (BLOB) object. Writing event data in binary format reduces the time it takes to write each record to the table. For this reason, UE tables are particularly useful where event monitor performance is important, which might be the case on highly I/O or CPU-bound systems.

However, because the event data is written in binary format, you cannot use SQL to extract legible data. You must perform post-processing on the UE table to extract the data stored in binary format. Another benefit of using UE tables is that you can have UE table data pruned automatically during post-processing. The EVMON_FORMAT_UE_TO_TABLES procedure has an option to delete data from the UE table after it has been successfully extracted.

To create an event monitor that writers to an unformatted event table, specify the WRITE TO UNFORMATTED EVENT TABLE clause in the CREATE EVENT MONITOR statement. Only one UE table is created per event monitor.

Files Some event monitors support sending their output directly to files maintained by the file system. This type of output is useful if you do not want the event monitor output to be subject to the additional processing time caused when being managed within the database, or if you want to look at the data while the database is offline. To create an event monitor that writes to files, specify the WRITE TO FILE clause in the CREATE EVENT MONITOR statement.

Named pipes
If you want to have an application process event data as it is generated, you can use a named pipe event monitor. These types of event monitors send their output directly to a named pipe so that the data can be used by another application immediately. This might be useful if you need to manipulate event data in real time.

To create an event monitor that writers to a named pipe, specify the WRITE TO PIPE clause in the CREATE EVENT MONITOR statement.

Depending on your needs, one type of event monitor output might be more appropriate than another. Table 82 provides an summary of when specific output types are particularly useful.

Table 82. Summary of different event monitor output types

Output type	Scenarios where this output type is useful
Regular tables	• When you want to examine monitoring data at a later point in time • In systems that are not approaching the maximum capacity for CPU, log file or disk storage • Where immediate access to data using SQL is desirable
Unformatted event (UE) tables	• When you want to examine monitoring data at a later point in time • In systems where event monitor performance is a priority, or where there are constraints on CPU, log file or disk usage • Where the added step of post-processing of data is not an issue
Files	• In systems where you do not want or need to manage monitor data as part of the database. (Eliminates the additional processing time of logging, inserts, maintaining consistency) • When you want to store the data outside of the database being monitored • When you want to examine the data offline at later point in time
Pipes	• Streaming event data to an application that processes it immediately. • When there is no need to access event data at a later point in time.

Not all event monitors support all output types. For example, only the unit of work, package cache and locking event monitor can produce a UE table. Table 83 on page 444 shows what output options are available for different types of event monitors:

Table 83. Output options for event monitors

Event monitor type	Regular table	Unformatted event table	File	Named pipe
Activity	Yes		Yes	Yes
Buffer pool	Yes		Yes	Yes
Change history	Yes			
Connections	Yes		Yes	Yes
Database	Yes		Yes	Yes
Deadlocks*(all variations)	Yes		Yes	Yes
Locking	Yes	Yes		
Package cache	Yes	Yes		
Statement	Yes		Yes	Yes
Statistics	Yes		Yes	Yes
Table space	Yes		Yes	Yes
Table	Yes		Yes	Yes
Threshold violations	Yes		Yes	Yes
Transaction*	Yes		Yes	Yes
Unit of work	Yes	Yes		
* Deprecated event monitor.				

Creating event monitors

You create different types of event monitors by using variations on the CREATE EVENT MONITOR statement. You can use the options for that statement to specify the type of data that event monitors collect and how the event monitors produce their output.

The sections that follow describe the different output options and how to create event monitors that produce these types of output.

Before you begin

Before creating an event monitor, it is important to understand the different options for the output that event monitors can produce. Most event monitors can produce output in at least two formats; some let you choose from up to four formats.

Procedure

To create an event monitor:

1. Determine what kind of event monitor you need.
2. Decide what type of output you want from the event monitor. Do you want data to be written to a regular table, an unformatted event table, a file, or a pipe?
3. Issue a CREATE EVENT MONITOR statement.
4. Optional: If the type of event monitor that you created requires activation, activate it by issuing the SET EVENT MONITOR STATE statement.

Creating event monitors that write to tables

To create an event monitor, use the CREATE EVENT MONITOR STATEMENT. There are different forms of this statement that you use, depending on the type of events that you intend to monitor.

Before you begin

- You need SQLADM or DBADM authority to create a table event monitor.
- The target table of a CREATE EVENT MONITOR statement - that is, the table to which the event monitor writes its output - must be a non-partitioned table.

About this task

The various options for table event monitors are set in the CREATE EVENT MONITOR statement. For further assistance in generating CREATE EVENT MONITOR SQL statements for write-to-table event monitors, you can use the **db2evtbl** command. Simply provide the name of the event monitor and the required event type (or types), and the CREATE EVENT MONITOR statement is generated, complete with listings of all the target tables. You can then copy the generated statement, make modifications, and then execute the statement from the command line processor.

Procedure

To create an event monitor that writes its output to a regular table, perform the following steps:

1. Formulate a CREATE EVENT MONITOR statement using the WRITE TO TABLE clause to indicate that event monitor data is to be collected in a table (or set of tables).

   ```
   CREATE EVENT MONITOR evmon-name FOR eventtype
      WRITE TO TABLE
   ```

   Where evmon-name is the name of the event monitor, and eventtype is one of the following values:
   - ACTIVITIES
   - BUFFERPOOLS
   - CHANGE HISTORY
   - CONNECTIONS
   - DATABASE
   - DEADLOCKS
   - LOCKING
   - PACKAGE CACHE
   - STATEMENTS
   - STATISTICS
   - TABLE
   - TABLESPACE
   - THRESHOLD VIOLATIONS
   - TRANSACTIONS
   - UNIT OF WORK

   For example, to create a unit of work event monitor called myevmon, use a statement like the one that follows:

   ```
   CREATE EVENT MONITOR myevmon FOR UNIT OF WORK
      WRITE TO TABLE
   ```

The preceding statement creates a unit of work event monitor that uses defaults for the logical groups of monitor elements collected, the corresponding output table names, and the target table spaces for the tables. For more information on these defaults, refer to the documentation for the appropriate CREATE EVENT MONITOR statement.

2. Optional: Specify the logical groups for which you want data collected. By default, event data is collected for all logical data groups for the event monitor type. If you want only data for selected logical groups collected, you can specify the names of the logical groups to include in the CREATE EVENT MONITOR statement. For example, with a locking event monitor, you might want to collect only the information associated with the LOCK and PARTICIPANT logical groups. To include only these logical groups, you can use a statement like the one that follows:

```
CREATE EVENT MONITOR mylocks FOR LOCKING
   WRITE TO TABLE
   LOCK, PARTICIPANTS
```

3. Optional: Specify the table names to use for the output tables. Unless you specify otherwise, default names are used for the tables for each logical group of monitor elements. The default name used is derived by concatenating the logical group name with the name of the event monitor. For example, for the locking event monitor created by the statement in the preceding step, the unqualified names for the tables produced are LOCK_MYLOCKS and PARTICIPANTS_MYLOCKS. To override the default names, include the table names to use when specifying the logical groups:

```
CREATE EVENT MONITOR mylocks FOR LOCKING
   WRITE TO TABLE
   LOCK(TABLE LOCKDATA), PARTICIPANTS(TABLE PARTICIP)
```

   In the preceding example, the names used for the tables for the LOCK and PARTICIPANTS logical groups are LOCKDATA_MYLOCKS and PARTICIP_MYLOCKS.

   You can also override the table space to be used for each table by including the name of the table space to use:

```
CREATE EVENT MONITOR mylocks FOR LOCKING
  WRITE TO TABLE
  LOCK(TABLE LOCKDATA IN EVMONSPACE), PARTICIPANTS(TABLE PARTICIP IN EVMONSPACE)
```

   In the preceding example, the EVMONSPACE table space is used for both output tables.

Additional options

Different event monitors provide different configuration options. For details on the options available for a specific type of event monitor, refer to the documentation for the CREATE EVENT MONITOR statement for the type of event monitor you want to use. The examples that follow show some of the configuration options you can choose for different event monitors:

Capturing multiple event types with a single event monitor

Some types[2] of event monitors can capture different types of events with a single event monitor. If you want to capture multiple types of events with this event monitor, specify additional values for `eventtype`, separated by a

[2] Event monitors for BUFFERPOOLS, CONNECTIONS, DATABASE, DEADLOCKS, STATEMENTS, TABLES, and TABLESPACES support this option.

comma. For example, you might want to combine bufferpool and table space monitoring in a single event monitor:

```
CREATE EVENT MONITOR myevmon FOR BUFFERPOOLS, TABLESPACES
                     WRITE TO TABLE
```

This event monitor will monitor for the BUFFERPOOL and TABLESPACE event types. Assuming that the previously listed statement was issued by the user `dbadmin`, the derived names and table spaces of the target tables are as follows:

- DBADMIN.BUFFERPOOL_MYEVMON
- DBADMIN.TABLESPACE_MYEVMON
- DBADMIN.CONTROL_,MYEVMON

Adjusting the size of event monitor output buffers

You can alter the size of the table event monitor buffers (in 4K pages) for some types[2] of event monitors by adjusting the BUFFERSIZE value. For example, in the following statement:

```
CREATE EVENT MONITOR myevmon FOR BUFFERPOOLS, TABLESPACES
                     WRITE TO TABLE BUFFERSIZE 8
```

8 is the combined capacity (in 4K pages) of the two event table buffers. This adds up to 32K of buffer space; 16K for each buffer.

The default size of each buffer is 4 pages (two 16K buffers are allocated). The minimum size is 1 page. The maximum size of the buffers is limited by the size of the monitor heap, because the buffers are allocated from that heap. For performance reasons, highly active event monitors should have larger buffers than relatively inactive event monitors.

Controlling whether event monitor output is blocked or non-blocked

Some event monitors[2] let you control how to proceed when event monitor output buffers are full. For blocked event monitors, each agent that generates an event will wait for the event buffers to be written to table if they are full. This can degrade database performance, as the suspended agent and any dependent agents cannot run until the buffers are clear. Use the BLOCKED clause to ensure no losses of event data:

```
CREATE EVENT MONITOR myevmon FOR BUFFERPOOLS, TABLESPACES
                     WRITE TO TABLE BUFFERSIZE 8 BLOCKED
```

If database performance is of greater importance than collecting every single event record, use non-blocked event monitors. In this case, each agent that generates an event will not wait for the event buffers to be written to table if they are full. As a result, non-blocked event monitors are subject to data loss on highly active systems. Use the NONBLOCKED clause to minimize the additional processing time caused by event monitoring:

```
CREATE EVENT MONITOR myevmon FOR BUFFERPOOLS, TABLESPACES
                     WRITE TO TABLE BUFFERSIZE 8 NONBLOCKED
```

Controlling what monitor elements for which data is collected

Which monitor elements to collect data for. If you are interested in only a few monitor elements, you can specify which ones you want to collect for some event monitors[2] by specifying the element name in the CREATE EVENT MONITOR statement:

```
CREATE EVENT MONITOR myevmon FOR DATABASE, BUFFERPOOLS, TABLESPACES
              WRITE TO TABLE DB, DBMEMUSE,
              BUFFERPOOL (EXCLUDES(db_path, files_closed)),
              TABLESPACE (INCLUDES
                (tablespace_name, direct_reads, direct_writes))
              BUFFERSIZE 8 NONBLOCKED
```

All the monitor elements for the DB and DBMEMUSE logical data groups are captured (this is the default behavior). For BUFFERPOOL, all monitor elements except **db_path** and **files_closed** are captured. And finally, for TABLESPACE, **tablespace_name**, **direct_reads** and **direct_writes** are the only monitor elements captured.

Setting a threshold for deactivating an event monitor based on table space used

All event monitors provide the option to specify how full the table space can get before the event monitor automatically deactivates:

```
CREATE EVENT MONITOR myevmon FOR BUFFERPOOLS, TABLESPACES
                     PCTDEACTIVATE 90
```

When the table space reaches 90% capacity, the myevmon event monitor automatically shuts off. The PCTDEACTIVATE clause can only be used for DMS table spaces. If the target table space has auto-resize enabled, set the PCTDEACTIVATE clause to 100.

What to do next

By default, event monitors that were introduced in Version 9.7 or later are created as AUTOSTART event monitors. They are activated automatically when the database is next activated, and on subsequent database activations thereafter. If you want to activate the event monitor immediately, before the next database activation, use the SET EVENT MONITOR STATE statement to manually start the event monitor. In addition for each of the locking, unit of work and package cache event monitors, you must also enable data collection.

Logical data groups and event monitor output tables

Monitor elements that are frequently used together are grouped into logical data groups. Event monitors that write to tables generally produce one output table for each logical data group of monitor elements that they capture. A complete list of all default target table names by event type is available at http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.admin.mon.doc/doc/r0059240.html.

Enabling event monitor data collection

Depending on the type of event monitor you are using, you might need to configure collection after you create the event monitor.

By default, some event monitors collect certain data immediately when activated. Other event monitors require that you explicitly configure data collection independently of creating the event monitor. These types of event monitors are sometimes referred to as *passive* event monitors.

Before you begin

All event monitors must be activated before any data is written its target output table or tables (regular or UE), file or pipe. Some event monitors are configured by default as AUTOSTART event monitors. This means they are activated automatically when the database is activated. Others are configured by default to required that you activate them manually. Either way, you can override the default startup options. However, to start an automatic event monitor after you create it, but before the next database activation, you must use the SET EVENT MONITOR STATE statement to activate it manually.

About this task

Some event monitors support the use of a WHERE clause on the CREATE or ALTER EVENT MONITOR statement to capture event information selectively. The following event monitors, however, provide the ability to control what event data is collected independently of the event monitor definition:

- Activities
- Change history
- Locking
- Statistics
- Unit of work

Some of the event monitors listed collect certain types of data by default after the event monitor is activated; others require that you explicitly enable data collection. Either way, you can enable data collection in one of two ways, depending on the scope of activities for which you want data collected:

All activities in the database

To have monitor data collected across all activities in the database, you modify the appropriate configuration parameter for the type of data you are interested in. For example, to have unit of work data collected for all units of work that run in the database, set **mon_uow_data** to BASE. In some cases, the default settings for configuration parameters are such that some type of data is always collected if there is an appropriate event monitor active to receive the date. For example, the default setting for **mon_req_metrics** is BASE; unless you override this setting, any active statistics or unit of work event monitor will record the values for the BASE set of request monitor elements.

Remember: Event monitors that support the use of the WHERE predicate collect only the data that satisfies the conditions specified in that predicate, regardless of the settings for any relevant configuration parameters.

Selected activities

Some event monitors - in particular, the workload management event monitors (threshold violations, statistics and activities) - provide the ability to control data collection for specific workload management objects. For example, you might choose to collect activity information for activities running in a specific service superclass. Configuring collection at this level generally involves adding a COLLECT clause to the CREATE or ALTER WORKLOAD (or SERVICE CLASS or WORK ACTION) statements to specify what type of information to collect for activities running under the auspices of that WLM object. For example, to enable the collection of extended statistics information for the service class `urgent`, you might use the following statement:

```
ALTER SERVICE CLASS urgent
     COLLECT AGGREGATE ACTIVITY DATA EXTENDED
```

Note: If a COLLECT clause is specified in a WLM CREATE or ALTER statement, the settings specified in the clause take precedence for that WLM object over any database-wide setting configured using a configuration parameter. For example, if **mon_req_metrics** is set to EXTENDED, and if workload payroll was configured to collect BASErequest metrics (for example, CREATE WORKLOAD payroll COLLECT REQUEST METRICS BASE), then extended request metrics are collected for all activities in the database *except* for the payroll workload.

Procedure

To enable collection of data for one of the types of event monitors shown at the beginning of this section, perform the following steps:

1. Determine what, if any data is already collected by default. The data you are interested in might be collected without you having to change any settings.
2. Decide on the scope of activities for which you want to collect data. Do you want to collect data for the entire database, or only for specific workloads, service class or work actions?
3. Decide what types of monitor elements you want to collect. Some event monitors support the collection of different types of monitor data, such as request monitor elements, activity data, and so on.
4. For the different sets of monitor data collected, decide the scope of data to be collected within each set. You generally have the choice of collecting no data (NONE), basic data (BASE), or extended data (EXTENDED). See to determine what data is collected for each setting.
5. Based on the decisions made in the preceding steps, configure data collection using either a configuration parameter or a COLLECT clause.
   a. To configure collection across the entire database, set the appropriate configuration parameter. For example, to enable the collection of lock wait information with history by the locking event monitor on the database SALES, run the following command.

      ```
      UPDATE DATABASE CONFIGURATION for SALES USING mon_lockwait HISTORY
      ```

   b. To configure collection for a specific workload, create or modify the workload, including the appropriate COLLECT clause. For example, to configure the collection of lock wait data with statement history for locks waiting longer than 5 seconds in the MANAGERS workload, run a statement like the one that follows:

      ```
         ALTER WORKLOAD MANAGERS
           COLLECT LOCK WAIT DATA FOR LOCKS WAITING MORE THAN 5 SECONDS
             WITH HISTORY
      ```

What to do next

Now that the event monitor is created and active, and data collection is enabled, run your applications or workload.

Methods for accessing event monitor information

Depending on the type of event monitor that you are using and the type of output it generates, there are different options for accessing and viewing event monitor data.

For example:

- Data produced by table event monitors can be queried directly using SQL.
- Data from event monitors that write to pipes can be viewed as it is produced.
- Data from file event monitors can be viewed by opening the output file after the event monitor is deactivated.

- Data from both file and pipe event monitors can also be formatted into a report using the **db2evmon** command.
- Data written to UE tables must be post-processed before it can be examined. UE event monitor data can be converted to tables or to XML, which makes it possible to query the data using SQL or XML query techniques. Alternatively, you can format the data in a UE table into a formatted report without going through a conversion process.

The sections that follow describe the different ways you can access information produced by event monitors.

Accessing event monitor data in regular tables

You can use SQL to directly access event monitor data that is written to regular relational tables.

Before you begin

Before accessing data, you must perform the following tasks:
- Create and activate the event monitor
- Enable data collection if required for the type of event monitor that you are using and the type of data that you want to collect
- Run the workload or applications for which you want to collect monitoring data

Optionally, depending on how you are using the event monitor data, deactivate data collection before you start examining the event data. If the event monitor remains active, it continues to write data to the output tables. Therefore, the results from one query might differ from the results that you obtain by running the same query later on.

About this task

Accessing event monitor data from relational tables involves using SQL to formulate queries to retrieve data from the tables produced by the event monitor.

Procedure

To retrieve information from the tables that are produced by an event monitor that writes to tables:
1. Formulate a SELECT statement to display the monitor element data you want to see. For example, to request lock data for the `payroll` workload from a locking event monitor named `mylocks`, you might use a query such as the following one:

   ```
   SELECT DISTINCT CAST(STMT_TEXT AS VARCHAR(25)) STMT, LP.PARTICIPANT_NO,
      VARCHAR(LP.APPL_NAME,10) APPL_NAME, LP.LOCK_MODE_REQUESTED,
        LP.PARTICIPANT_TYPE
   FROM LOCK_PARTICIPANT_ACTIVITIES_LOCK_MYLOCKS AS LPA
   JOIN LOCK_PARTICIPANTS_LOCK_MYLOCKS AS LP
      ON LPA.EVENT_ID = LP.EVENT_ID
      WHERE LP.WORKLOAD_NAME = 'PAYROLL'
   ```

   In this example, data from the LOCK_PARTICIPANTS table from the event monitor `mylocks` is joined with information from the LOCK_PARTICIPANTS_ACTIVITIES table to return the following results.
2. Run the SQL statement.

Results

```
STMT                      PARTICIPANT_NO APPL_NAME  LOCK_WAIT_VAL
------------------------- -------------- ---------- --------------------
select * from staff                    2 db2bp                         0
select * from staff                    1 db2bp                      1000

LOCK_MODE_REQUESTED  PARTICIPANT_TYPE
-------------------- ----------------
                   0 OWNER
                   1 REQUESTER


  2 record(s) selected.
```

Methods for accessing information in unformatted event tables

There are different ways to access the information in unformatted event (UE) tables. You can generate a text report intended to be read. Alternatively, you can extract the data into relational tables or XML; this approach lets you query the data using SQL or pureXML.

Event monitors that write to UE tables write event data in a binary format. You can access this data using the db2evmonfmt command or routines provided for this purpose.

With the db2evmonfmt command you can:

- select events of interest based on the following attributes: event ID, event type, time period, application, workload, or service class.
- choose whether to receive the output in the form of a text report or a formatted XML document.
- completely control the output format by creating your own XSLT style sheets instead of using the ones provided with db2evmonfmt.

You can also extract data from an unformatted event table using the following routines:

- EVMON_FORMAT_UE_TO_XML - extracts data from an unformatted event table into an XML document.
- EVMON_FORMAT_UE_TO_TABLES - extracts data from an unformatted event table into a set of relational tables.

With these two routines, you can use a SELECT statement to specify the exact rows from the unformatted event table that you want to extract.

Routines for extracting data from unformatted event tables

If you want to perform queries on the data collected by an event monitor that writes to a unformatted event (UE) table, you must first extract the data from UE table using one of the two routines provided for this purpose.

The EVMON_FORMAT_UE_TO_TABLES procedure extracts data from the UE table to create relational tables. The EVMON_FORMAT_UE_TO_XML table function creates an XML document.

EVMON_FORMAT_UE_TO_TABLES

The EVMON_FORMAT_UE_TO_TABLES procedure examines the UE table produced by an event monitor, and extracts the data it contains into relational tables that you can query. The number of tables produced depends on the type of event monitor; and the logical data groups for which that event monitor collects data. Generally speaking, the data from each logical data group is written to a separate table. For example, the package cache event monitor collects event data from three logical data groups: pkgcache and pkgcache_metrics, and pkgcache_stmt_args. Thus, three tables are produced by EVMON_FORMAT_UE_TO_TABLES.

Note: EVMON_FORMAT_UE_TO_TABLES does not create a table for the control logical data group.

In addition to creating relational tables from UE tables, as of Version 10.1 the EVMON_FORMAT_UE_TO_TABLES procedure provides the capability to prune data from UE tables. When you use the `PRUNE_UE_TABLES` option for EVMON_FORMAT_UE_TO_TABLES, data that is successfully inserted into relational tables is deleted from the unformated event (UE) table.

EVMON_FORMAT_UE_TO_XML

The EVMON_FORMAT_UE_TO_XML table function examines the UE table produced by an event monitor, and extracts the data it contains into an XML document. This document can then be queried as often as needed using pureXML.

Notes:

- This table function works similarly to the **db2evmonfmt** utility when that utility is used with the `-fxml` option. The differences between using EVMON_FORMAT_UE_TO_XML instead of **db2evmonfmt** are as follows:
  - EVMON_FORMAT_UE_TO_XML is a table function. As such, it is invoked as part of an SQL statement. **db2evmonfmt** runs as a separate utility.
  - EVMON_FORMAT_UE_TO_XML lets you specify a SELECT statement with a WHERE clause to filter events from the UE table. **db2evmonfmt** has only limited capabilities for filtering event data.
- The output XML document from EVMON_FORMAT_UE_TO_XML can be formatted by **db2evmonfmt** to create a flat text file.

With both routines, you must include a SELECT statement in the call to the routine to specify conditions for which data to extract.

Pruning data from UE tables

If you use the EVMON_FORMAT_UE_TO_TABLES procedure to extract data from UE tables, you can use the PRUNE_UE_TABLE option to remove data that you no longer need.

Before you begin

Before you can extract data from a UE table, you must have created, activated, and enabled data collection for an event monitor that writes to a UE table.

About this task

In addition to the performance advantages that UE tables offer, using UE tables as output for an event monitor lets you take advantage of the automatic pruning feature of the EVMON_FORMAT_UE_TO_TABLES procedure. When you use this procedure, any data that is extracted from the UE table and written to a regular table can be automatically removed from the UE table. This procedure makes it easier to manage a UE table. For example, assume that you want to use a unit of work event monitor to capture information to generate daily reports for accounting purposes, such as charging departments for CPU time that is used by an application or query. In that case, you might want to prune that data after producing the reports.

Procedure

To extract and then prune data from a UE table:

Issue an SQL statement that calls the EVMON_FORMAT_UE_TO_TABLES procedure with the PRUNE_UE_TABLE option to extract data into a regular table. For example, if you have a unit of work event monitor called TRACKWORK, you might create a statement such as the one that follows:

```
CALL EVMON_FORMAT_UE_TO_TABLES
   ('UOW', NULL, NULL, NULL, NULL, NULL, 'PRUNE_UE_TABLE', -1,
      'SELECT * FROM TRACKWORK')
```

All event data is copied from the UE table to the UOW_EVENT_TRACKWORK and UOW_METRICS_TRACKWORK tables. In addition, all records that were copied are removed from the UE table.

Formatting file or pipe event monitor output from a command line

The output of a file or pipe event monitor is a binary stream of logical data groupings. You can format this data stream from a command line by using the **db2evmon** command.

This productivity tool reads in event records from an event monitor's files or pipe, then writes them to the screen (standard output).

Before you begin

No authorization is required unless you are connecting to the database, in which case one of the following authorities is required:

- SYSADM
- SYSCTRL
- SYSMAINT
- DBADM

About this task

You can indicate which event monitor output to format by either providing the path of the event files, or providing the name of the database and the event monitor name.

Procedure

To format event monitor output:

- Specify the directory containing the event monitor files:

  ```
  db2evmon -path '/tmp/dlevents'
  ```

  `/tmp/dlevents` represents a (UNIX) path.
- Specify the database and event monitor name:

  ```
  db2evmon -db 'sample' -evm 'dlmon'
  ```

  `sample` represents the database the event monitor belongs to.

dlmon represents an event monitor.

Altering an event monitor

You can add one or more logical data groups to the set of logical data groups that the event monitor collects by using the ALTER EVENT MONITOR statement.

About this task

When you create an event monitor that writes to tables, by default, all logical data groups of monitor elements that are associated with that event monitor are captured. However, if you include the names of logical data groups in the CREATE EVENT MONITOR statement, only those groups are captured. For example, you might create an activities event monitor that captures data only from the event_activity and event_activity_metrics logical data groups, as shown in the following example:

```
CREATE EVENT MONITOR myacts FOR ACTIVITIES
   WRITE TO TABLE
   event_activity, event_activity_metrics
```

The preceding DDL statement creates an event monitor that writes to two tables: ACTIVITY_myacts and ACTIVITY_METRICS_myacts.

Restrictions

You can use the ALTER EVENT MONITOR statement only to add logical data groups to an event monitor. You cannot remove a logical data group. You also cannot change the name, the target table space, or the value for PCTDEACTIVATE that is associated with the table that is used to capture the data in monitor elements that belong to a data group.

Procedure

To add additional logical data groups to an event monitor:

1. Decide which logical data group you want to add. Using the preceding example of a locking event monitor where only two logical data groups are being captured, assume that you want to add the event_activitystmt and event_activityvals logical data groups.
2. Formulate an ALTER EVENT MONITOR statement to add these new logical data groups.
   ```
   ALTER EVENT MONITOR mylacts
      ADD LOGICAL GROUP event_activitystmt
      ADD LOGICAL GROUP event_activityvals
   ```
3. Execute the statement.

Results

When the ALTER EVENT MONITOR statement completes execution, two additional tables are created for the event monitor myacts:

ACTIVITYSTMT_myacts

ACTIVITYVALS_myacts

The next time the event monitor is activated, these tables are populated with data from their corresponding logical data groups.

Remember: If you add new logical data groups to an event monitor, any data that existed for the logical data groups that were originally part of the table will not have any corresponding rows in the tables for the newly added logical group. Adjust your queries as needed, or consider pruning old data from the table after adding the logical groups.

Example

A database administrator creates a locking event monitor called `mylocks` by using the following SQL statement:

```
CREATE EVENT MONITOR mylocks FOR LOCKING WRITE TO TABLE LOCK, LOCK_PARTICIPANTS
```

This statement collects information for monitor elements in the lock and lock_participants logical data groups. The tables to which the monitor element data is written are created with the default table names LOCK_MYLOCKS and LOCK_PARTICIPANTS_MYLOCKS.

Later on, the database administrator decides that she wants to collect information in the LOCK_PARTICIPANT_ACTIVITIES logical data group. She uses the following statement to modify the event monitor:

```
ALTER EVENT MONITOR mylocks ADD LOGICAL GROUP LOCK_PARTICIPANT_ACTIVITIES
```

This statement causes the monitor elements in the lock_participant_activities to be collected in addition to the other elements that already were collected. This new set of monitor elements are written to the table LOCK_PARTICIPANT_ACTIVITIES_MYLOCKS.

Later, the database administrator decides that she also needs the data from the control logical data group. However, she wants this data to be written to a table with a name other than the default name, and to a table space other than the default table space. She uses the following statement:

```
ALTER EVENT MONITOR mylocks ADD LOGICAL GROUP CONTROL TABLE ctl_mylocks IN mytbsp3
```

This statement adds the control logical data group to the output of the event monitor. This statement adds the control logical data group to the output of the event monitor. The data is written to the CTL_MYLOCKS table, and the table is written to the table space mytbsp3, instead of the default table space.

Reports generated using the MONREPORT module

The MONREPORT module generates text reports of monitoring data that you can use to troubleshoot SQL performance problems.

You can generate the following reports using the MONREPORT module:

Table 84. List of reports generated using the MONREPORT module

Report Name	Procedure to create report	Main data source / table functions
Summary report	MONREPORT.DBSUMMARY	MON_GET_SERVICE_SUBCLASS and selected details from MON_GET_CONNECTION and MON_GET_WORKLOAD
Connection report	MONREPORT.CONNECTION	MON_GET_CONNECTION
Current Applications report	MONREPORT.CURRENTAPPS	Includes fields from MON_GET_CONNECTION, MON_GET_UNIT_OF_WORK, WLM_GET_SERVICE_CLASS_AGENTS, WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES
Current SQL report	MONREPORT.CURRENTSQL	MON_GET_PKG_CACHE_STMT (For the **executable_id** obtained from the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function.)
Package Cache report	MONREPORT.PKGCACHE	MON_GET_PKG_CACHE_STMT

Table 84. List of reports generated using the MONREPORT module (continued)

Report Name	Procedure to create report	Main data source / table functions
Current Lock Wait report	MONREPORT.LOCKWAIT	Most data from MON_GET_APPL_LOCKWAIT; additional data from MON_GET_CONNECTION, WLM_GET_SERVICE_CLASS_AGENTS, WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES, MON_GET_PKG_CACHE_STMT, MON_GET_TABLE

Most reports start with a summary section that provides one line of key information for each item in the report. For example, the Connection report contains a one-line summary of each connection. The main body of the report consists of a detailed section for each item in the summary.

Each metric in the report is labeled with the underlying monitor element name (for example: CLIENT_IDLE_WAIT_TIME = 44). To determine what the metric represents, search the Information Center for the monitor element name.

You can customize the reports generated by the MONREPORT module. The MONREPORT module is implemented entirely using SQL and you can obtain the module code from the database catalog and create a customized version.

Reports for initial diagnosis

An important use of these reports is to troubleshoot SQL performance slowdowns. Each report is designed to answer certain diagnosis questions. Some reports support initial diagnosis, while others support subsequent detailed diagnosis of particular types of problems.

Initial diagnosis involves:

- Determining the problem category, by narrowing the problem down to the aspect or stage of processing that has slowed down.
- Identifying the SQL statements involved in the problem and collecting information about the SQL statements for further analysis.

Table 85. MONREPORT module reports suitable for initial diagnosis

Procedure name	Information provided and usage
MONREPORT.DBSUMMARY Part 1: System Performance	Part 1 of the Summary report provides monitor data for most aspects of processing aggregated across the entire database. This information is useful for answering questions about the aspect or stage of processing that has slowed down. For example: • Is the problem inside or outside the data server? • Is there a computing resource bottleneck? • Are requests in a wait state? If so, for what resource? • Is the slowdown located in a particular data server processing component?

Table 85. MONREPORT module reports suitable for initial diagnosis (continued)

Procedure name	Information provided and usage
MONREPORT.DBSUMMARY Part 2: Application Performance	Part 2 of the Summary report provides key performance indicators for each connection, workload, and service class. This information is useful for answering questions about the scope of application requests involved in the slowdown. For example: • Is this slowdown a general system slowdown that affects much or all the workload? • Is this slowdown limited to SQL statements issued from a particular source such as particular connections, DB2 workloads or DB2 service classes?
MONREPORT.DBSUMMARY Part 3: Member level information	Part 3 of the Summary report provides key performance indicators for each member. This information is useful for determining whether the slowdown is isolated to one or a few members.
MONREPORT.CURRENTSQL	The current SQL report provides information about statements that are currently running, in the form of several lists of the top *N* activities. The statements are ranked by different metrics: processing resource, rows processed, direct reads and direct writes. This information is useful for determining whether the slowdown is isolated to one or a few SQL statements. If the slowdown is isolated to one or a few SQL statements, those statements are likely to appear in this report of top statements.
MONREPORT.PKGCACHE	The package cache report provides information about statements that have run recently and are stored in the package cache. This report shows several summaries, each listing the top *N* activities. The activities are ranked by the following monitor elements: • CPU • wait time • rows processed • num_coord_exec_with_metrics - Number of executions by coordinator agent with metrics monitor element (if no member was specified) or num_exec_with_metrics - Number of executions with metrics collected monitor element (if a member was specified) • I/O wait time This report contains a summary for each of these metrics as well as a report for each execution. This information is useful for determining whether the slowdown is isolated to one or a few SQL statements. If so, those statements are likely to appear at the top in this report. The information per execution can help identify the most costly statements while the information summed across executions can help identify statements with the most impact on the system cumulatively considering both the statement cost and frequency of execution.

Table 85. MONREPORT module reports suitable for initial diagnosis (continued)

Procedure name	Information provided and usage
MONREPORT.CURRENTAPPS	The current applications report show the current processing state for units of work, agents, and activities. The report starts with a summary section showing the number of current connections and activities, as well as a series of summaries, such as the summary of current units of work by workload occurrence state. The body of the report consists of one section for each connection that provides the details of the connection. This information is useful for viewing all the work currently running on the system. This allows you to check for patterns that might identify the problem category.

Reports for detailed diagnosis

After completing the initial diagnosis, you might need to pursue a specialized or detailed set of troubleshooting analyses for the problem category you identified during the initial diagnosis phase.

Table 86. MONREPORT module reports suitable for detailed diagnosis

Procedure name	Information provided and usage
MONREPORT.CONNECTION	If the MONREPORT.DBSUMMARY report showed that the slowdown is limited to SQL statements issued from a particular connection, then you can view detailed information about the affected connection. This report contains the same metrics as Part 1 of the MONREPORT.DBSUMMARY report, but it presents this information for each connection.
MONREPORT.LOCKWAIT	If the reports viewed during the initial diagnosis suggest there is a lock wait problem, then you can view detailed information about each lock wait currently in progress. This information includes lock holder and lock requester details, as well as characteristics of the lock held and the lock requested.

Chapter 29. Monitoring DB2 workload management environments

The third domain of workload management is monitoring, which must be performed on an ongoing basis.

The primary purpose of monitoring is to validate the health and efficiency of your system and the individual workloads running on it. Using table functions, you can access real-time operational data such as a list of running workload occurrences and the activities running in a service class or average response times. Using event monitors you can capture detailed activity information and aggregate activity statistics for historical analysis.

Looking at aggregate information should usually be the first step when you build a monitoring strategy. Aggregates give a good picture of overall data server activity and are also cheaper because you do not have to collect information on every activity in which you might be interested. You can collect more detailed information as you understand the scope of your monitoring needs.

Typical monitoring tasks you can perform are:

- Analyzing the workload on your system to help design your initial DB2 workload management configuration.
- Tracking and investigating the behavior of your system by obtaining types of operational information that permit you to:
  - Analyze system performance degradation
  - Diagnose activities that are taking too long to complete
  - Investigate agent contention
  - Isolate poorly performing queries

  Information is available for activities, service classes, workloads, work classes, threshold queues, and threshold violations.
- Exercising control over the execution environment by canceling queued activities that you expect will cause problems or cancel running activities that you have diagnosed as negatively impacting the system.

Real-time monitoring with table functions

Real-time monitoring data includes information about work currently running on the system, statistics, and metrics for work that has been performed on the system that can help you to determine usage patterns and resource allocation and identify problem areas. You use DB2 table functions to obtain this operational information.

Table functions with names that begin with *WLM_* are DB2 workload management table functions. These table functions provide access to a set of data relevant to managing your workload, such as workload management statistics, as a virtual DB2 table against which you can issue a SELECT statement. This enables you to write applications to query data and analyze it as if it were in a physical table on the data server. The DB2 workload management table functions are qualified with the SYSPROC schema name.

Table functions with names that begin with *MON_* are monitoring metrics functions. Monitoring metrics provide monitoring data about the health of and query performance on your DB2 data server, which can then be used as input to a 3rd party tool or in conjunction with additional scripting you provide to analyze the metrics returned. Only those monitoring metrics functions that are relevant for DB2 workload management are included here. The monitor metrics table functions are similar to the workload management statistics table functions. Both return elements describing work that has taken place on the system. The key differences between these monitoring metrics table functions and the DB2 workload management table functions are:

- The DB2 workload management table functions provide data that is more statistical in nature, such as computed values like averages, high watermarks, standard deviations, etc. In contrast, the monitoring metrics table functions provide a much more complete set of raw monitoring data.
- The data reported by the DB2 statistics functions is reset when data is sent to a statistics event monitor. This resetting of data is necessary to make values such as high watermarks meaningful over a specific collection interval. Data reported by the monitoring metrics functions is also captured by a statistics event monitor, but is never reset. The data reported by monitoring interfaces accumulates from the time a database is activated until the time it is deactivated.

Statistical information

General statistical information is also available for a number of different objects. You can use this statistical information for a number of different purposes, such as for verifying that changes to your DB2 workload management configuration have had the expected effect. If you create a new work class to classify READ activities, for example, you can verify that READ activities are being classified under the new work class correctly. You can also use table functions to quickly recognize certain problems with the system. For example, you can use table functions to determine an acceptable value for the average activity lifetime and recognize when this value exceeds its usual range, possibly indicating a problem that requires further investigation.

Statistics are useful only if the time period during which they are collected is meaningful. Collecting statistics over a very long time, and for any length of time using the WLM_COLLECT_STATS stored procedure, might be less useful if it becomes difficult to identify changes to trends or problem areas because there is too much old data. Thus, you can reset statistics at any time.

Because of the default workload and default user service classes, monitoring capabilities exist from the moment that you install the DB2 data server. These can help you to start identifying sources of activities that you can use to create workloads and the service classes to which you can assign them.

Example: Using DB2 workload management table functions

A large amount of data is available through DB2 workload management real-time monitoring. The example in this topic shows how you might start using the information.

In this situation, only the default workload and service class are in place. Use this example to understand how you can use the table functions to understand what, exactly, is running on the data server. Follow these steps:

1. Use the Service Superclass Statistics table function to show all of the service superclasses. After you install or upgrade to DB2 9.5 or later, three default

superclasses are defined: one for maintenance activities, one for system activities, and one for user activities. SYSDEFAULTUSERCLASS is the service class of interest.

```
SELECT VARCHAR(SERVICE_SUPERCLASS_NAME,30) AS SUPERCLASS
   FROM TABLE(WLM_GET_SERVICE_SUPERCLASS_STATS('',-1)) AS T

SUPERCLASS
------------------------------
SYSDEFAULTSYSTEMCLASS
SYSDEFAULTMAINTENANCECLASS
SYSDEFAULTUSERCLASS

  3 record(s) selected.
```

2. Use the Service Subclass Statistics table function to show statistics for all the service subclasses of the SYSDEFAULTUSERCLASS superclass. For each service subclass you can see the current volume of requests that are being processed, the number of activities that have completed execution, and the overall distribution of activities across members (possibly indicating a problem if the distribution is uneven). You can optionally obtain additional statistics including the average lifetime for activities, the average amount of time activities spend queued, and so on. You can obtain optional statistics for a service subclass by specifying the COLLECT AGGREGATE ACTIVITY DATA keyword on the ALTER SERVICE CLASS statement to enable aggregate activity statistics collection.

```
SELECT VARCHAR(SERVICE_SUPERCLASS_NAME, 20) AS SUPERCLASS,
   VARCHAR(SERVICE_SUBCLASS_NAME, 20) AS SUBCLASS,
   COORD_ACT_COMPLETED_TOTAL,
   COORD_ACT_ABORTED_TOTAL,
   COORD_ACT_REJECTED_TOTAL,
   CONCURRENT_ACT_TOP
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS(
   'SYSDEFAULTUSERCLASS', 'SYSDEFAULTSUBCLASS', -1))
   AS T
```

```
SUPERCLASS           SUBCLASS             COORD_ACT_COMPLETED_TOTAL COORD_ACT_ABORTED_TOTAL COORD_ACT_REJECTED_TOTAL CONCURRENT_ACT_TOP
-------------------- -------------------- ------------------------- ----------------------- ------------------------ ------------------
SYSDEFAULTUSERCLASS  SYSDEFAULTSUBCLASS                           2                       0                        0                  1

  1 record(s) selected.
```

3. For a given service subclass, use the Workload Occurrence Information table function to list the occurrences of a workload that are mapped to the service subclass. The table function displays all of the connection attributes, which you can use to identify the source of the activities. This information can be quite useful in determining custom workload definitions in the future. For example, perhaps a specific workload occurrence listed here has a large volume of work from an application as shown by the activities completed counter.

```
SELECT APPLICATION_HANDLE,
   VARCHAR(WORKLOAD_NAME, 30) AS WORKLOAD,
   VARCHAR(SESSION_AUTH_ID, 20) AS SESSION_AUTH_ID,
   VARCHAR(APPLICATION_NAME, 20) AS APPL_NAME
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES(
   'SYSDEFAULTUSERCLASS', 'SYSDEFAULTSUBCLASS', -1))
   AS T
```

```
APPLICATION_HANDLE   WORKLOAD                       SESSION_AUTH_ID      APPL_NAME
-------------------- ------------------------------ -------------------- --------------------
                 431 SYSDEFAULTUSERWORKLOAD         SWALKTY              db2bp

  1 record(s) selected.
```

   a. For that application, use the Workload Occurrence Activities Information table function to show the current activities across database members that were created from the application's connection. You can use this information for a number of purposes, including identifying activities that might be causing problems on the data server.

```
SELECT APPLICATION_HANDLE,
   LOCAL_START_TIME,
   UOW_ID,
   ACTIVITY_ID,
   ACTIVITY_TYPE
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(431,-1)) AS T

APPLICATION_HANDLE   LOCAL_START_TIME           UOW_ID      ACTIVITY_ID ACTIVITY_TYPE
-------------------- -------------------------- ----------- ----------- --------------------------------
                 431 2008-06-17-12.49.46.854259          11           1 READ_DML

  1 record(s) selected
```

b. For each activity, retrieve more detailed information by using the Activity Details table function. The data might show that some SQL statements are returning huge numbers of rows, that some activities have been idle for a long time, or that some queries are running that have an extremely large estimated cost. In situations such as these, it might make sense to define some thresholds to identify and prevent potentially damaging behavior in the future.

```
SELECT VARCHAR(NAME, 20) AS NAME,
   VARCHAR(VALUE, 40) AS VALUE
FROM TABLE(WLM_GET_ACTIVITY_DETAILS(431,11,1,-1))
   AS T WHERE NAME IN ('UOW_ID', 'ACTIVITY_ID', 'STMT_TEXT')

NAME                 VALUE
-------------------- ----------------------------------------
UOW_ID               1
ACTIVITY_ID          1
STMT_TEXT            select * from syscat.tables

  3 record(s) selected.
```

Example: Monitoring current system behavior at different levels

DB2 workload management provides a number of table functions that you can use to obtain data about your workload management configuration.

Installing DB2 Version 9.5 or later creates a set of default workloads and service classes. Before deciding how to implement your own DB2 workload management solution, you can use the table functions to observe work being performed in the system in terms of the default workload occurrences, service classes, and activities.

You can start by obtaining the list of workload occurrences in a service class. To do this, use the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function. In the following example, an empty string is passed for *service_superclass_name* and *service_subclass_name*, and -2 (a wildcard character) is passed for *member*:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(CHAR(COORD_MEMBER),1,4) AS COORDMEMB,
       SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHNDL,
       SUBSTR(CHAR(WORKLOAD_NAME),1,22) AS WORKLOAD_NAME,
       SUBSTR(CHAR(WORKLOAD_OCCURRENCE_ID),1,6) AS WLO_ID
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('', '', -2)) AS SCINFO
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, MEMB, APPHNDL, WORKLOAD_NAME, WLO_ID
```

Assume that the system has four database members and that there are two applications performing activities on the database when you issue the query. The results would resemble the following ones:

```
SUPERCLASS_NAME     SUBCLASS_NAME      MEMB COORDMEMB APPHNDL WORKLOAD_NAME          WLO_ID
------------------- ------------------ ---- --------- ------- -----------------------------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    0         1       SYSDEFAULTUSERWORKLOAD 1
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    0         2       SYSDEFAULTUSERWORKLOAD 2
```

The results indicate that both workload occurrences were assigned to the SYSDEFAULTUSERWORKLOAD workload. The results also show that both workload occurrences were assigned to the SYSDEFAULTSUBCLASS service subclass in the SYSDEFAULTUSERCLASS service superclass and that both workload occurrences are from the same coordinator member (member 0).

Next, you can also use the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function again to determine the connection attributes of the two workload occurrences:

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHNDL,
       SUBSTR(CHAR(WORKLOAD_NAME),1,22) AS WORKLOAD_NAME,
       SUBSTR(CHAR(WORKLOAD_OCCURRENCE_ID),1,6) AS WLO_ID,
       SUBSTR(CHAR(SYSTEM_AUTH_ID),1,9) AS SYSAUTHID,
       SUBSTR(CHAR(APPLICATION_NAME),1,15) AS APPLNAME
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('', '', 0)) AS SCINFO
ORDER BY APPHNDL, WORKLOAD_NAME, WLO_ID

APPHNDL WORKLOAD_NAME          WLO_ID SYSAUTHID APPLNAME
------- ---------------------- ------ --------- ---------------
1       SYSDEFAULTUSERWORKLOAD 1      LYNN      accountspay
2       SYSDEFAULTUSERWORKLOAD 2      KATE      businessobjects
```

Then, you can use the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function to show the current activities of one of the workload occurrences:

```
SELECT SUBSTR(CHAR(COORD_MEMBER),1,5) AS COORD,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(CHAR(UOW_ID),1,5) AS UOWID,
       SUBSTR(CHAR(ACTIVITY_ID),1,5) AS ACTID,
       SUBSTR(CHAR(PARENT_UOW_ID),1,8) AS PARUOWID,
       SUBSTR(CHAR(PARENT_ACTIVITY_ID),1,8) AS PARACTID,
       SUBSTR(ACTIVITY_TYPE,1,9) AS ACTTYPE,
       SUBSTR(CHAR(NESTING_LEVEL),1,7) AS NESTING
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(1, -2)) AS WLOACTS
ORDER BY MEMB, UOWID, ACTID

COORD MEMB UOWID ACTID PARUOWID PARACTID ACTTYPE  NESTING
----- ---- ----- ----- -------- -------- -------- -------
0     0    1     3     -        -        CALL     0
0     0    1     5     1        3        READ_DML 1
0     1    1     5     -        -        READ_DML 1
0     2    1     5     -        -        READ_DML 1
0     3    1     5     -        -        READ_DML 1
```

The query results show that workload occurrence 1 is running two activities. One activity is a stored procedure (indicated by the activity type of CALL), and the other activity is a DML activity that performs a read (for example, a SELECT statement). The DML activity is nested in the stored procedure call. You can tell that the DML activity is nested because the parent unit of work identifier and parent activity identifier of the DML activity match the unit of work identifier and the activity identifier of the CALL activity. You can also tell that the DML activity is executing on database members 0, 1, 2, and 3. The parent identifier information is available only on the coordinator member.

You can obtain more information about an individual activity that is currently running by using the MON_GET_ACTIVITY_DETAILS table function. This table function returns an XML document where the elements in the document describe the activity. In this example, the XMLTABLE function is used to return a result table from the XML output.

```
SELECT D.APP_HANDLE,
       D.MEMBER,
       D.COORD_MEMBER,
       D.LOCAL_START_TIME,
       D.UOW_ID,
       D.ACTIVITY_ID,
       D.PARENT_UOW_ID,
       D.PARENT_ACTIVITY_ID,
       D.ACTIVITY_TYPE,
       D.NESTING_LEVEL,
       D.INVOCATION_ID,
       D.ROUTINE_ID
FROM TABLE(MON_GET_ACTIVITY_DETAILS(65592, 1, 1, -2)) AS ACTDETAILS,
XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
           '$details/db2_activity_details' PASSING XMLPARSE(DOCUMENT
    ACTDETAILS.DETAILS) as "details"
COLUMNS "APP_HANDLE"         BIGINT    PATH 'application_handle',
        "MEMBER"             BIGINT    PATH 'member',
        "COORD_MEMBER"       BIGINT    PATH 'coord_member',
        "LOCAL_START_TIME"   VARCHAR(26) PATH 'local_start_time',
        "UOW_ID"             BIGINT    PATH 'uow_id',
        "ACTIVITY_ID"        BIGINT    PATH 'activity_id',
        "PARENT_UOW_ID"      BIGINT    PATH 'parent_uow_id',
        "PARENT_ACTIVITY_ID" BIGINT    PATH 'parent_activity_id',
        "ACTIVITY_TYPE"      VARCHAR(10) PATH 'activity_type',
        "NESTING_LEVEL"      BIGINT    PATH 'nesting_level',
        "INVOCATION_ID"      BIGINT    PATH 'invocation_id',
        "ROUTINE_ID"         BIGINT    PATH 'routine_id'
) AS D;
```

```
APP_HANDLE MEMBER COORD_MEMBER LOCAL_START_TIME           UOW_ID ACTIVITY_ID
---------- ------ ------------ -------------------------- ------ -----------
65592      1      1            2009-04-07-18.39.42.549197 1      1
65592      0      1            2009-04-07-18.39.42.552763 1      1

 PARENT_UOW_ID PARENT_ACTIVITY_ID ACTIVITY_TYPE NESTING_LEVEL INVOCATION_ID ROUTINE_ID
 ------------- ------------------ ------------- ------------- ------------- ----------
 -             -                  READ_DML      0             0             0
 -             -                  READ_DML      0             0             0

  2 record(s) selected.
```

Note: The query results have been divided in two parts for readability purposes.

The table functions mentioned previously provide a high-level description of work that is running in the system. The information that these table functions provide regarding the status of the work is limited to an activity state such as EXECUTING. If you want to probe further to discover what exactly is occurring in a service class at a point in time, you can run the WLM_GET_SERVICE_CLASS_AGENTS table function.

In the following example, WLM_GET_SERVICE_CLASS_AGENTS is called by passing 1 for *application_handle* and -2 (a wildcard character) for *member*:

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(CHAR(AGENT_TID),1,9) AS AGENT_TID,
       SUBSTR(AGENT_TYPE,1,11) AS AGENTTYPE,
       SUBSTR(AGENT_STATE,1,10) AS AGENTSTATE,
       SUBSTR(REQUEST_TYPE,1,14) AS REQTYPE,
       SUBSTR(CHAR(UOW_ID),1,6) AS UOW_ID,
       SUBSTR(CHAR(ACTIVITY_ID),1,6) AS ACT_ID
```

```
FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS('', '', 1, -2)) AS SCDETAILS
ORDER BY APPHANDLE, MEMB, AGENT_TID

APPHANDLE MEMB AGENT_TID AGENTTYPE   AGENTSTATE REQTYPE      UOW_ID ACT_ID
--------- ---- --------- ----------- ---------- --------------------------
1         0    3         COORDINATOR ACTIVE     FETCH             1      5
1         0    4         PDBSUBAGENT ACTIVE     SUBSECTION:1      1      5
1         1    2         PDBSUBAGENT ACTIVE     SUBSECTION:2      1      5
```

The results show a coordinator agent and a subagent on member 0 and a subagent on member 1 operating on behalf of an activity with a unit of work identifier of 1 and an activity identifier of 5. The coordinator agent information indicates that the request is a fetch request.

Historical monitoring with WLM event monitors

DB2 workload management uses event monitors to capture information that might be of use in the future or for historical analysis.

Three event monitors are available for you to use. Each event monitor serves a different purpose:

Activity event monitor

This monitor captures information about individual activities in a service class, workload, or work class or activities that violated a threshold. The amount of data that is captured for each activity is configurable and should be considered when you determine the amount of disk space and the length of time required to keep the monitor data. A common use for activity data is to use it as input to tools such as **db2advis** or to use access plans (from the explain utility) to help determine table, column, and index usage for a set of queries.

You can collect information about an activity by specifying COLLECT ACTIVITY DATA for the service class, workload, or work action to which such an activity belongs or a threshold that might be violated by such an activity. The information is collected when the activity completes, regardless of whether the activity completes successfully.

Note that if an activities event monitor is active when the database deactivates, any backlogged activity records in the queue are discarded. To ensure that you obtain all activities event monitor records and that none are discarded, explicitly deactivate the activities event monitor first before deactivating the database. When an activities event monitor is explicitly deactivated, all backlogged activity records in the queue are processed before the event monitor deactivates.

Threshold violations event monitor

This monitor captures information when a threshold is violated. It indicates what threshold was violated, the activity that caused the violation, and what action was taken when it occurred.

If you specify COLLECT ACTIVITY DATA for the threshold and an activities event monitor is created and active, information is also collected about activities that violate the threshold, but this information is collected when the activity ends (either successfully or unsuccessfully).

You can obtain details about a threshold by querying the SYSCAT.THRESHOLDS view.

Statistics event monitor

This monitor serves as a low-overhead alternative to capturing detailed activity information by collecting aggregate data (for example, the number of activities completed and average execution time). Aggregate data includes histograms for a number of activity measurements including lifetime, queue time, execution time and estimated cost. You can use histograms to understand the distribution of values, identify outliers, and compute additional statistics such as averages and standard deviations. For example, histograms can help you understand the variation in lifetime that users experience. The average life time alone does not reflect what a user experiences if there is a high degree of variability.See “Collecting workload management statistics using a statistics event monitor” for a description of how to send statistics to the event monitor.

The following figure shows the different monitoring options available to access workload information: table functions to access real-time statistics, and activity details and historical information captured as efficient aggregates or as details about individual activities:

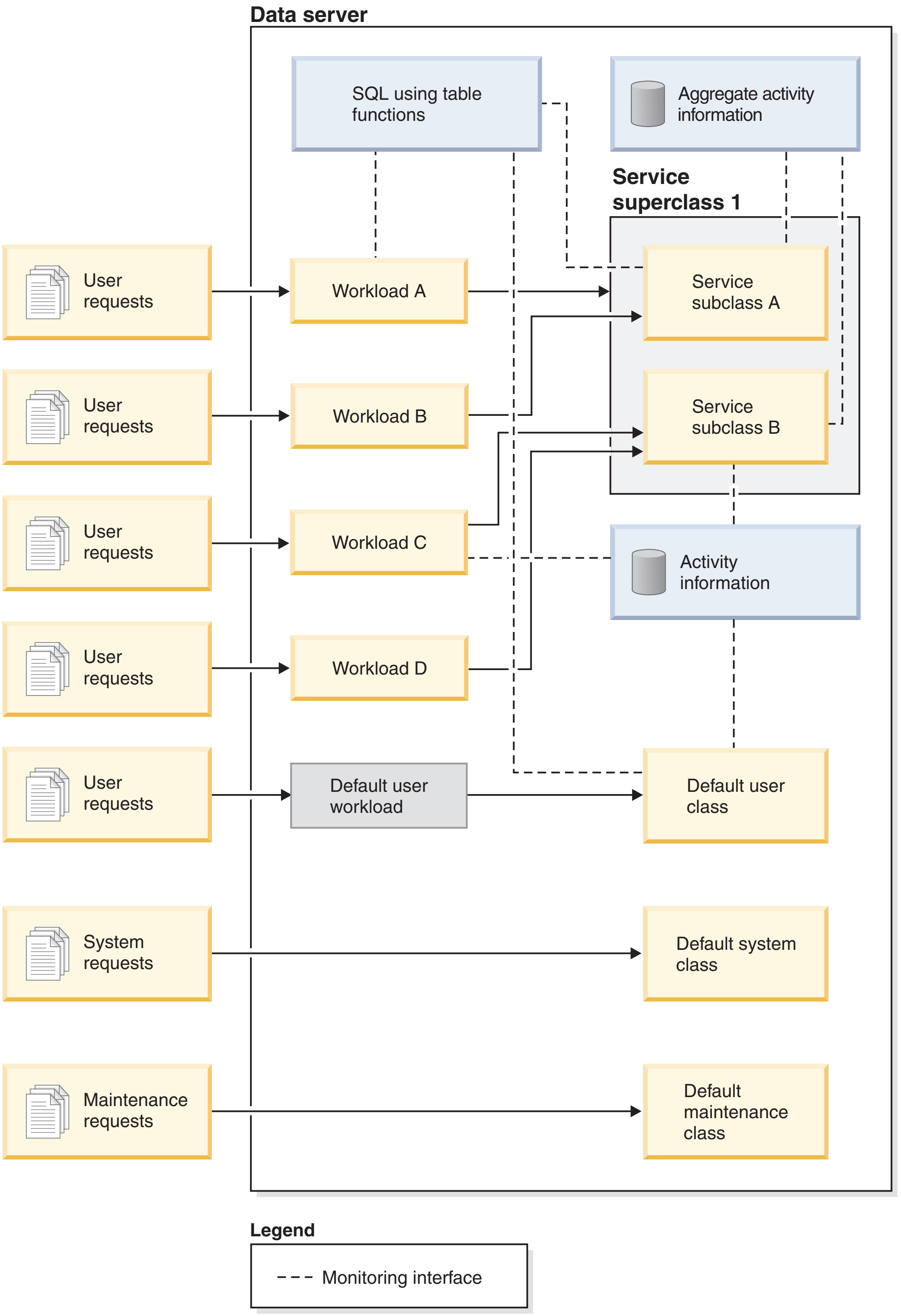


Figure 39. Workload management with monitoring

Unlike statement, connection, and transaction event monitors, the activity, statistics, and threshold violations event monitors do not have event conditions (that is, conditions specified on the WHERE keyword of the CREATE EVENT MONITOR statement). Instead, these event monitors rely on the attributes of service classes, workloads, work classes, and thresholds to determine whether these objects send their activity information or aggregate information to these monitors.

Typically, event monitors write data to either tables or files. You need to prune these tables or files periodically because they are not automatically pruned.

You can use the **wlmevmon.ddl** script in the sqllib/misc directory to create and enable three event monitors called DB2ACTIVITIES, DB2STATISTICS, and DB2THRESHOLDVIOLATIONS. If necessary, modify the script to change the table space or other parameters.

Example

Example: Identify queries with a large estimated cost using the statistics event monitor: You suspect that your database workload occasionally includes large, expensive queries, possibly due to the poor optimization of the queries themselves. You want to identify these queries so that you can prevent them from consuming excessive resources on your system, with a long-term goal of perhaps rewriting some of the queries to improve performance. The statistics event monitor provides you with a low-overhead way to measure the estimated cost of your queries which you can then use to determine what the maximum acceptable estimated cost for a query on your data server should be. A query that is poorly optimized is typically distinguished by a large estimated cost that is many times larger than the estimated cost of most other queries.

To get started, you need to create and activate a statistics event monitor and to start collecting extended aggregate activity data for the service class where the queries run:

```
CREATE EVENT MONITOR DB2STATISTICS
  FOR STATISTICS WRITE TO TABLE

SET EVENT MONITOR DB2STATISTICS STATE 1
```

In this example, all queries run in the SYSDEFAULTSUBCLASS subclass of the SYSDEFAULTUSERCLASS service class, which you can alter to collect the required data:

```
ALTER SERVICE CLASS SYSDEFAULTSUBCLASS UNDER SYSDEFAULTUSERCLASS
   COLLECT AGGREGATE ACTIVITY DATA EXTENDED
```

A full day of work might provide a reasonable approximation of the range of queries your data server typically processes. At the end of the day, you can copy the statistics collected from memory to the statistics event monitor by running the WLM_COLLECT_STATS stored procedure:

```
CALL WLM_COLLECT_STATS()
```

Included with the different statistics written to the event monitor tables are the estimated cost statistics of queries. To see them, you can query the service class statistics table SCSTATS_DB2STATISTICS:

```
SELECT STATISTICS_TIMESTAMP,
       COORD_ACT_EST_COST_AVG,
       COST_ESTIMATE_TOP
FROM SCSTATS_DB2STATISTICS
WHERE SERVICE_SUPERCLASS_NAME = 'SYSDEFAULTUSERCLASS'
  AND SERVICE_SUBCLASS_NAME = 'SYSDEFAULTSUBCLASS'

STATISTICS_TIMESTAMP       COORD_ACT_EST_COST_AVG COST_ESTIMATE_TOP
-------------------------- ---------------------- --------------------
2008-09-03-09.49.04.455979                 169440             13246445

  1 record(s) selected.
```

The output shows that the average query has an estimated cost in the range of hundreds of thousands of timerons, and that the largest queries have estimated costs larger than ten million timerons. You can confirm that queries of ten million or more timerons are outliers by looking at the estimated cost histogram, which was generated at the same time that the average and high watermarks shown in the output were written to the event monitor table. You can look at the histogram by querying the HISTOGRAMBIN_DB2STATISTICS table as follows:

```
SELECT STATISTICS_TIMESTAMP,
       TOP,
       NUMBER_IN_BIN
FROM HISTOGRAMBIN_DB2STATISTICS HIST,
     SYSCAT.SERVICECLASSES SC
WHERE HIST.SERVICE_CLASS_ID = SC.SERVICECLASSID
  AND SC.PARENTSERVICECLASSNAME = 'SYSDEFAULTUSERCLASS'
  AND SC.SERVICECLASSNAME = 'SYSDEFAULTSUBCLASS'
  AND HISTOGRAM_TYPE = 'COORDACTESTCOST'"
```

```
STATISTICS_TIMESTAMP       TOP                  NUMBER_IN_BIN
-------------------------- -------------------- --------------------
2008-09-03-09.49.04.455979                    1                    0
2008-09-03-09.49.04.455979                    2                    0
2008-09-03-09.49.04.455979                    3                    0
2008-09-03-09.49.04.455979                    5                    0
2008-09-03-09.49.04.455979                    8                    0
2008-09-03-09.49.04.455979                   12                    1
2008-09-03-09.49.04.455979                   19                    0
2008-09-03-09.49.04.455979                   29                    0
2008-09-03-09.49.04.455979                   44                    2
2008-09-03-09.49.04.455979                   68                    5
2008-09-03-09.49.04.455979                  103                   22
2008-09-03-09.49.04.455979                  158                   14
2008-09-03-09.49.04.455979                  241                   54
2008-09-03-09.49.04.455979                  369                    2
2008-09-03-09.49.04.455979                  562                  142
2008-09-03-09.49.04.455979                  858                   21
2008-09-03-09.49.04.455979                 1309                  123
2008-09-03-09.49.04.455979                 1997                  512
2008-09-03-09.49.04.455979                 3046                  643
2008-09-03-09.49.04.455979                 4647                  201
2008-09-03-09.49.04.455979                 7089                  875
2008-09-03-09.49.04.455979                10813                 1445
2008-09-03-09.49.04.455979                16493                 5386
2008-09-03-09.49.04.455979                25157                 2409
2008-09-03-09.49.04.455979                38373                 8940
2008-09-03-09.49.04.455979                58532                 9820
2008-09-03-09.49.04.455979                89280                 2149
2008-09-03-09.49.04.455979               136181                  798
2008-09-03-09.49.04.455979               207720                 2411
2008-09-03-09.49.04.455979               316840                14989
2008-09-03-09.49.04.455979               483283                 9831
2008-09-03-09.49.04.455979               737162                 1451
2008-09-03-09.49.04.455979              1124409                  213
2008-09-03-09.49.04.455979              1715085                   24
2008-09-03-09.49.04.455979              2616055                    1
2008-09-03-09.49.04.455979              3990325                    0
2008-09-03-09.49.04.455979              6086529                    0
2008-09-03-09.49.04.455979              9283913                    0
2008-09-03-09.49.04.455979             14160950                    3
2008-09-03-09.49.04.455979             21600000                    0
2008-09-03-09.49.04.455979                   -1                    0
```

In the histogram, the value in the number_in_bin column for queries whose top is greater than 2616055 is zero until top reaches 14160950, where the number_in_bin becomes 3. These three queries are outliers and can be controlled with an ESTIMATEDSQLCOST threshold set to trigger if the estimated cost of a query exceeds 10 million timerons which you can use to prevent such activities from executing and to monitor them more closely.

Example: Using the threshold violations event monitor: To control activities of a certain estimated cost, you want to define an ESTIMATEDSQLCOST threshold on your workload that applies only to that subset of your total workload exceeding a certain estimated cost. Having looked at the estimated cost histogram, you determined that activities with an estimated cost in the range of 0 to under 3 million timerons occur frequently and that activities with an estimated cost over 10 million timerons occur rarely (perhaps only a few times a day and perhaps always due to some flaw in the query, such as the use of a Cartesian join).

To verify that a threshold of 10 million timerons is effective in stopping those few activities a day that should not be allowed to run, you can create and activate a threshold event monitor:

```
CREATE THRESHOLD TH1
  FOR DATABASE ACTIVITIES
  ENFORCEMENT DATABASE
  WHEN ESTIMATEDSQLCOST > 10000000
  STOP EXECUTION

CREATE EVENT MONITOR DB2THRESHOLDVIOLATIONS
  FOR THRESHOLD VIOLATIONS
  WRITE TO TABLE

SET EVENT MONITOR DB2THRESHOLDVIOLATIONS STATE 1
```

After the end of the day, you can see what threshold violations occurred by querying the threshold violations table:

```
SELECT THRESHOLDID,
       SUBSTR(THRESHOLD_PREDICATE, 1, 20) PREDICATE,
       TIME_OF_VIOLATION,
       THRESHOLD_MAXVALUE,
       THRESHOLD_ACTION
FROM THRESHOLDVIOLATIONS_DB2THRESHOLDVIOLATIONS
ORDER BY TIME_OF_VIOLATION, THRESHOLDID
```

```
THRESHOLDID PREDICATE            TIME_OF_VIOLATION          THRESHOLD_MAXVALUE   THRESHOLD_ACTION
----------- -------------------- -------------------------- -------------------- ----------------
          1 EstimatedSQLCost     2008-09-02-22.39.10.000000             10000000 Stop

  1 record(s) selected.
```

Example: Using the activity event monitor

The previous example showed how you can collect threshold information in an event monitor table to confirm that activities with a large estimated cost are being prevented from executing by a threshold. After seeing these threshold violations, you want to determine what the SQL statement texts producing these large queries are, so that you can use the explain facility to determine if an index is needed on the tables being queried.

Collecting this additional information requires creating and activating an activity event monitor and altering the threshold to turn on activity collection with details:

```
CREATE EVENT MONITOR DB2ACTIVITIES
  FOR ACTIVITIES WRITE TO TABLE

SET EVENT MONITOR DB2ACTIVITIES STATE 1
```

```
ALTER THRESHOLD TH1
  WHEN EXCEEDED
  COLLECT ACTIVITY DATA WITH DETAILS
```

When you query the threshold violations table again after another business day has passed, you can perform a join with the ACTIVITYSTMT_DB2ACTIVITIES table to see the SQL statement text of any activity that violated the threshold:

```
SELECT THRESHOLDID,
       SUBSTR(THRESHOLD_PREDICATE, 1, 20) PREDICATE,
       TIME_OF_VIOLATION,
       SUBSTR(STMT_TEXT,1,70) STMT_TEXT
FROM THRESHOLDVIOLATIONS_DB2THRESHOLDVIOLATIONS TV,
     ACTIVITYSTMT_DB2ACTIVITIES A
WHERE TV.APPL_ID = A.APPL_ID
  AND TV.UOW_ID = A.UOW_ID
  AND TV.ACTIVITY_ID = A.ACTIVITY_ID
```

```
THRESHOLDID PREDICATE            TIME_OF_VIOLATION          STMT_TEXT
----------- -------------------- -------------------------- ----------------------------------------------------------------------
          1 EstimatedSQLCost     2008-09-02-23.04.49.000000 select count(*) from syscat.tables,syscat.tables,syscat.tables

  1 record(s) selected.
```

DB2 workload management monitoring data

Monitoring data is available from workloads, service subclasses and service superclasses, work classes, and threshold queues. You can use this data to diagnose and correct problems and for performance tuning.

Workload monitoring data

The following figure shows the monitoring information that is available for workloads. You can collect workload statistics and information about activities that run in the workloads using event monitors. For workloads, you can also obtain aggregate activity statistics. You can access workload statistics and information about workload occurrences in real time using table functions.

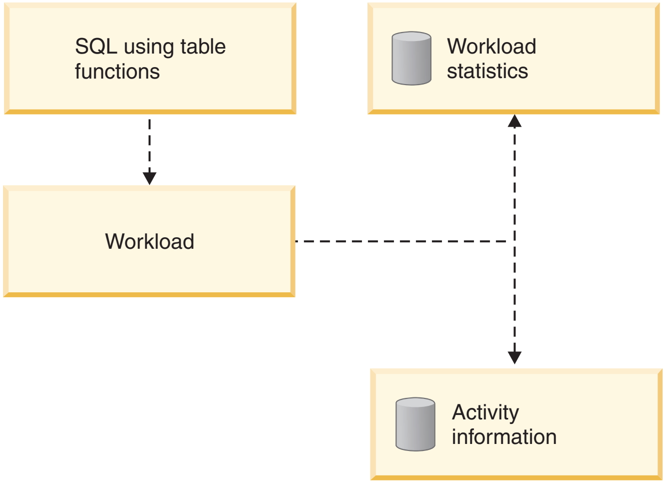


Figure 40. Monitoring data that is available for workloads

Service class monitoring data

The following figure shows the monitoring information that is available for service classes. You can collect statistics for service subclasses and service superclasses. For service subclasses, you can also obtain aggregate activity and request statistics, and information about activities that run in the service subclass. You can access service superclass and service subclass statistics and information about agents running in a particular service class in real time using table functions.

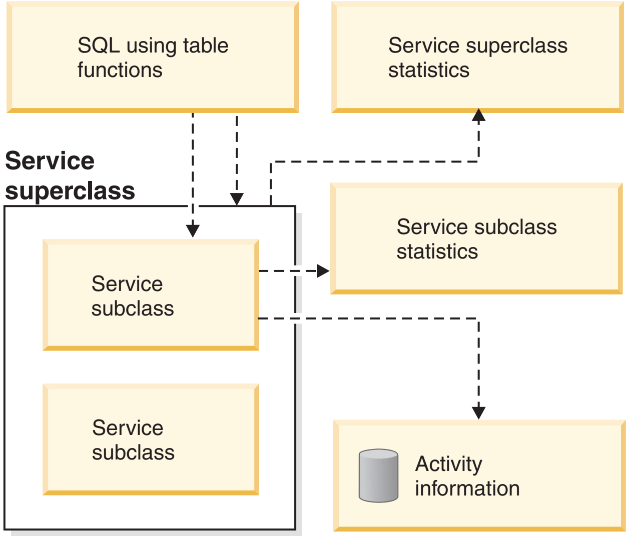

SQL using table functions
Service superclass statistics
Service superclass
Service subclass statistics
Service subclass
Service subclass
Activity information

Figure 41. Monitoring data that is available for service classes

Work class monitoring data

The following figure shows the monitoring information that is available for work classes. You can collect work class statistics and information about activities that are associated with a particular work class. You can access work class statistics in real time using table functions.

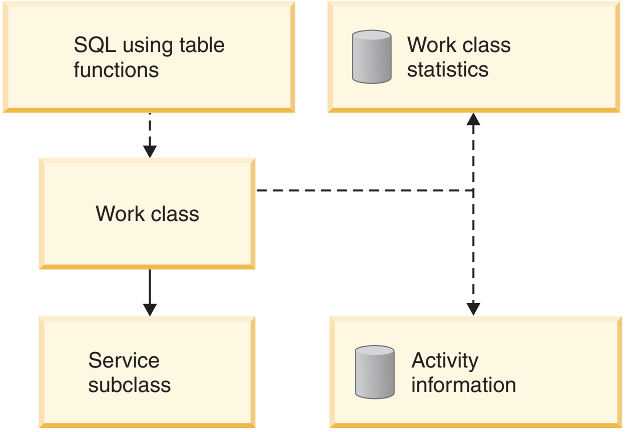


Figure 42. Monitoring data that is available for work classes

Threshold monitoring data

The following figure shows the monitoring information that is available for thresholds. You can obtain information about threshold violations, the activities that caused the threshold violations, and queuing statistics (for queuing thresholds). You can access queuing threshold statistics in real time using table functions.

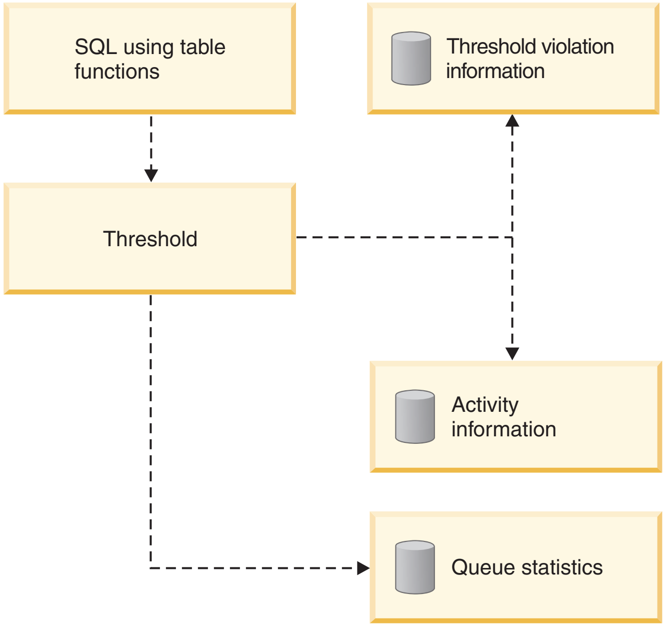


Figure 43. Monitoring data that is available for thresholds

DB2 workload management stored procedures

You can use stored procedures for canceling an activity, capturing details about an activity, resetting the statistics on DB2 workload management objects, and setting client information at the data server.

The following stored procedures are available for use with DB2 workload management:

WLM_CANCEL_ACTIVITY(*application_handle***,** *uow_id***,** *activity_id***)**
Use this stored procedure to cancel a running or queued activity. You identify the activity by its application handle, unit of work identifier, and activity identifier. You can cancel any type of activity. The application with the cancelled activity receives the error SQL4725N.

WLM_CAPTURE_ACTIVITY_IN_PROGRESS(*application_handle***,** *uow_id***,** *activity_id***)**
Use this stored procedure to send information about an individual activity that is currently executing to the activities event monitor. This stored procedure sends the information immediately, rather than waiting until the activity completes.

WLM_COLLECT_STATS()
Use this stored procedure to collect and reset statistics for DB2 workload management objects. All statistics tracked for service classes, workloads, threshold queues, and work action sets are sent to the active statistics event monitor (if one exists) and reset. If there is no active statistics event monitor, the statistics are only reset, but not collected.

WLM_SET_CLIENT_INFO(*client_userid***,** *client_wrkstnname***,***client_applname***,** *client_acctstr***,***client_workload***)**
Use this procedure to set the client information attributes used at the data server to record the identity of the application or end-user currently using

the connection. In cases where middleware exists between applications or users and your data server, use the WLM_SET_CLIENT_INFO procedure to set distinguishing connection attributes explicitly.

Workload management table functions and snapshot monitor integration

You can use DB2 workload management table functions with the snapshot monitor table functions when performing problem determination or performance tuning.

The DB2 workload management table functions and the snapshot monitor table functions share the following fields. You can perform joins on these fields to derive data that you need to perform diagnostic and performance-tuning activities. Note that, unlike the snapshot table functions, the WLM table functions do not get their information from the snapshot monitor, so that the information available in the WLM table functions is not available from the snapshot monitor.

Table 87. Fields shared between the DB2 workload management and snapshot monitor table functions

Workload management table function field	Snapshot monitor table function field
agent_tid	**agent_pid**
application_handle	**agent_id** **agent_id_holding_lock**
session_auth_id	**session_auth_id**
member	**node_number**
utility_id	**utility_id**
workload_id	**workload_id**

As an example of a reason to use a join between different table functions, assume that you want to obtain basic information about all of the utilities running in the BATCH service superclass. You might issue the following query:

```
SELECT SUBSTR(UTILITY_TYPE,1,4) TYPE,
       UTILITY_PRIORITY PRIORITY,
       SUBSTR(UTILITY_DESCRIPTION,1,12) DESCRIPTION,
       SUBSTR(UTILITY_DBNAME,1,8) DBNAME,
       UTILITY_STATE STATE,
       SUBSTR(UTILITY_INVOKER_TYPE,1,7) INVOKER,
       SUBSTR(CHAR(WLM.MEMBER),1,4) MEMB,
       SUBSTR(CLASSES.PARENTSERVICECLASSNAME,1,19) SUPERCLASS_NAME,
       SUBSTR(CLASSES.SERVICECLASSNAME,1,18) SUBCLASS_NAME
FROM SYSIBMADM.SNAPUTIL SNAP,
     TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(CAST(NULL AS BIGINT), -2)) WLM,
     SYSCAT.SERVICECLASSES CLASSES
WHERE SNAP.UTILITY_ID = WLM.UTILITY_ID
  AND WLM.SERVICE_CLASS_ID = CLASSES.SERVICECLASSID
  AND CLASSES.SERVICECLASSNAME = 'SYSDEFAULTSUBCLASS'
  AND CLASSES.PARENTSERVICECLASSNAME = 'BATCH'
ORDER BY WLM.MEMBER
```

The output might resemble the following output:

```
TYPE PRIORITY DESCRIPTION  DBNAME STATE   INVOKER MEMB SUPERCLASS_NAME SUBCLASS_NAME
---- -------- ------------ ------ ------- ------- ---- --------------- ------------------
LOAD        - OFFLINE LOAD SAMPLE EXECUTE USER    1    BATCH           SYSDEFAULTSUBCLASS
LOAD        - OFFLINE LOAD SAMPLE EXECUTE USER    1    BATCH           SYSDEFAULTSUBCLASS
LOAD        - OFFLINE LOAD SAMPLE EXECUTE USER    1    BATCH           SYSDEFAULTSUBCLASS
LOAD        - OFFLINE LOAD SAMPLE EXECUTE USER    2    BATCH           SYSDEFAULTSUBCLASS
LOAD        - OFFLINE LOAD SAMPLE EXECUTE USER    2    BATCH           SYSDEFAULTSUBCLASS
LOAD        - OFFLINE LOAD SAMPLE EXECUTE USER    2    BATCH           SYSDEFAULTSUBCLASS
LOAD        - OFFLINE LOAD SAMPLE EXECUTE USER    3    BATCH           SYSDEFAULTSUBCLASS
LOAD        - OFFLINE LOAD SAMPLE EXECUTE USER    3    BATCH           SYSDEFAULTSUBCLASS
LOAD        - OFFLINE LOAD SAMPLE EXECUTE USER    3    BATCH           SYSDEFAULTSUBCLASS
```

Monitoring metrics for DB2 workload management

Monitoring metrics provide data about the health of and query performance on your DB2 data server, which can then be used as input to a 3rd party tool or in conjunction with additional scripting you provide to analyze the metrics returned.

Metrics are maintained for a number of DB2 database objects. These metrics reside in memory and can be viewed in real-time using DB2 monitoring metrics table functions, or the metrics can be collected and sent to an event monitor where they can be viewed later for historical analysis.

Monitoring metrics for activities

You can obtain monitoring metrics for activities using:

- The activities event monitor (the ACTIVITYMETRICS table or the DETAILS_XML column of the ACTIVITY table)
- The MON_GET_ACTIVITY_DETAILS table function

Monitoring metrics for activities are controlled by the **mon_act_metrics** database configuration parameter and the COLLECT ACTIVITY METRICS clause on a workload. Metrics will be collected for an activity, if the database configuration parameter is set to a value other than NONE or if the activity is submitted by a connection that is associated with a workload which has a COLLECT ACTIVITY METRICS setting other than NONE.

You can use workload-level controls to achieve better monitoring granularity, if you do not want to collect metrics for all activities. If activity metrics collection is enabled at the database level (enabled by default), then metrics are collected for all activities, regardless of the setting at the workload level.

See the monitoring documentation for more details.

System-level monitoring metrics

You can obtain system-level monitoring metrics aggregated by service classes and workloads using:

- The statistics event monitor (the **details_xml** and **metrics** monitor elements in the EVENT_SCSTATS and EVENT_WLSTATS logical data groups or individual monitor elements in the EVENT_SCMETRICS and EVENT_WLMETRICS logical data groups)
- The MON_GET_SERVICE_SUBCLASS, MON_GET_SERVICE_SUBCLASS_DETAILS, MON_GET_WORKLOAD and MON_GET_WORKLOAD_DETAILS table functions

Monitoring metrics for requests to the data server, including those requests that are part of an activity, are controlled by the **mon_req_metrics** database configuration parameter and the COLLECT REQUEST METRICS clause on a service superclass. Metrics will be collected for a request, if the database configuration parameter is set to a value other than NONE or if the request is submitted by a connection that mapped to a subclass under a superclass which has a COLLECT COLLECT REQUEST METRICS setting other than NONE.

You can use service superclass-level controls to achieve better monitoring granularity, if you do not want to collect metrics for all requests. If request metrics collection is enabled at the database level (enabled by default), then metrics are collected for all requests, regardless of the setting at the service superclass level.

See the monitoring documentation for more details.

Monitoring threshold violations

When a DB2 workload manager threshold is violated, a threshold violation record is written to the active THRESHOLD VIOLATIONS event monitor, if one exists.

About this task

The threshold violation record includes the following information:

- A description of the threshold that was violated (the identifier, maximum value, and so on).
- An identification of the activity that violated the threshold, including the identifier of the application that submitted the activity, the unique activity identifier, and the unit of work identifier.
- The time that the threshold was violated.
- The action that was taken. The action indicates whether the activity that violated the threshold was permitted to continue or was stopped. If the activity was stopped, the application that submitted the activity will have received an SQL4712N error.

When a threshold violation occurs for a threshold that has a REMAP ACTIVITY action defined for it, a threshold violation record is optional. Whether or not a threshold violation record is recorded is determined by the NO EVENT MONITOR RECORD or LOG EVENT MONITOR RECORD clause of your CREATE THRESHOLD statement.

You can optionally have detailed activity information (including statement text) written to an active activities event monitor if the threshold violation is caused by an activity. The activity information is written when the activity completes, not when the threshold is violated. Specify that activity information should be collected when a threshold is violated by using the COLLECT ACTIVITY DATA keyword on either the CREATE or ALTER threshold or work action set statements.

Procedure

To monitor threshold violations:

1. Use the CREATE EVENT MONITOR statement to create an event monitor of type THRESHOLD VIOLATIONS. For example:
   ```
   CREATE EVENT MONITOR VIOLATIONS FOR THRESHOLD VIOLATIONS WRITE TO TABLE
   ```
2. Use the COMMIT statement to commit your changes.
3. Use the SET EVENT MONITOR STATE statement to activate the event monitor. Instead of using the SET EVENT MONITOR STATE statement, you can use the AUTOSTART default for the THRESHOLD VIOLATIONS event monitor to have it activated the next time that the database is activated. If you want to define multiple THRESHOLD VIOLATIONS event monitors, you should not use the AUTOSTART option.
4. Use the COMMIT statement to commit your changes.

Note: If you create any thresholds, you should create and activate a threshold violations event monitor so you can monitor any threshold violations that occur. A threshold violations event monitor does not have any impact unless thresholds are violated.

Example

This example shows how you can determine what remappings of a particular activity occurred as the result of a threshold violation that included a REMAP ACTIVITY action. To find the activities that were remapped, use a statement like the following:

```
SELECT VARCHAR(APPL_ID, 30) AS APPLID,
   UOW_ID,
   ACTIVITY_ID,
   VARCHAR(T.PARENTSERVICECLASSNAME,20) AS SERVICE_SUPERCLASS,
   VARCHAR(T.SERVICECLASSNAME,20) AS FROM_SERVICE_SUBCLASS,
   VARCHAR(S.SERVICECLASSNAME,20) AS TO_SERVICE_SUBCLASS
FROM THRESHOLDVIOLATIONS_TH1,
   SYSCAT.SERVICECLASSES AS T,
   SYSCAT.SERVICECLASSES AS S
WHERE SOURCE_SERVICE_CLASS_ID = T.SERVICECLASSID AND
      DESTINATION_SERVICE_CLASS_ID = S.SERVICECLASSID AND
      THRESHOLD_ACTION = 'REMAP'
ORDER BY APPLID, ACTIVITY_ID, UOW_ID, TIME_OF_VIOLATION ASC;
```

In this example, two remappings occurred for the activity submitted by the application with the ID *N0.swalkty.080613140844 which is identified by activity ID 1 and unit of work (UOW) ID 1:

```
APPLID                         UOW_ID      ACTIVITY_ID          SERVICE_SUPERCLASS   FROM_SERVICE_SUBCLASS TO_SERVICE_SUBCLASS
------------------------------ ----------- -------------------- -------------------- --------------------- --------------------
*N0.swalkty.080613140844                 1                    1 WORK                 HIGH                  MED
*N0.swalkty.080613140844                 1                    1 WORK                 MED                   LOW

  2 record(s) selected.
```

The output is ordered by the time of threshold violation and shows that the activity was remapped twice after it started executing. Although not shown in the output, the initial service subclass the activity was mapped to is likely a high priority service subclass, typical of a three-tiered configuration that permits shorter running queries to complete more quickly. Because the activity did not complete quickly enough in the high priority service subclass, it violated a threshold and was remapped to a medium priority service subclass, and then remapped again to a low priority service subclass after a second threshold violation later on.

Collecting data for individual activities

You can use an ACTIVITIES event monitor to collect data for individual activities that run in your system. The data collected includes items such as statement text and compilation environment, and can be used to investigate and diagnose problems, and as input to other tools (for example, the Design Advisor).

About this task

You can collect information about individual activities for service subclasses, workloads, work classes (through work actions), and threshold violations. You enable activity collection using the COLLECT ACTIVITY DATA keyword of the CREATE and ALTER statements for these DB2 workload management objects. When an activity completes, information about the activity is sent to the active ACTIVITIES event monitor if:

- The activity was submitted by an application that is mapped to a workload for which COLLECT ACTIVITY DATA is specified, or
- The activity runs in a service subclass for which COLLECT ACTIVITY DATA is specified, or
- The activity has a COLLECT ACTIVITY DATA work action applied to it, or
- The activity violates a threshold that was defined with the COLLECT ACTIVITY DATA action

You can also use the WLM_SET_CONN_ENV procedure to turn on activity collection for your own application's connection before executing the user's query, then execute the user's query, then use WLM_SET_CONN_ENV to turn off activity collection for your application's own connection. Assuming that you have created and activated an activity event monitor, the application could look something like the following:

```
call WLM_SET_CONN_ENV(cast (NULL as bigint),
  '<collectactdata>WITHOUT DETAILS</collectactdata>')
```

... execute user's query ...

```
call WLM_SET_CONN_ENV(cast(NULL as bigint), '<collectactdata>NONE</collectactdata>')
```

The COLLECT ACTIVITY DATA keyword also controls the amount of information that is sent to the ACTIVITIES event monitor. If the keyword specifies WITH DETAILS, statement information (such as statement text) is collected. If the keyword specifies WITH DETAILS AND VALUES, data values are collected as well.

An activity might have multiple COLLECT ACTIVITY DATA keywords applied to it. For example, the activity might run in a service subclass for which COLLECT ACTIVITY DATA is specified, and while executing it might violate a threshold that has the COLLECT ACTIVITY DATA action. In this situation, the activity is only collected once. The COLLECT keyword that specifies the largest amount of information to be collected is applied to the activity. For example, if both COLLECT ACTIVITY DATA WITHOUT DETAILS and COLLECT ACTIVITY DATA WITH DETAILS are applied to an activity, the activity is collected with detailed information.

If the ON ALL DATABASE MEMBERS keywords are used with the COLLECT ACTIVITY DATA clause, an activity record will be captured on each member where the activity executes in a multimember database environment. Activity event monitor records are written when the last agent working on the activity at that member completes execution. Depending on the sequencing of events in a section, it is possible for agents to start and stop working on an activity at a member several times, causing multiple activity records to be captured at that member for the same query. The total work done by the activity on that member is the aggregate of the metrics for each record that is captured for the activity on the member.

Procedure

To enable collection of activities for a given DB2 workload management object:

1. Use the CREATE EVENT MONITOR statement to create an ACTIVITIES event monitor.
2. Use the COMMIT statement to commit your changes.

3. Use the SET EVENT MONITOR STATE statement to activate the event monitor. Instead of using the SET EVENT MONITOR STATE statement, you can use the AUTOSTART default for the ACTIVITIES event monitor to have it activated the next time that the database is activated. If you want to define multiple ACTIVITIES event monitors, you should not use the AUTOSTART option.
4. Use the COMMIT statement to commit your changes.
5. Identify the objects for which you want to collect activities by using the ALTER SERVICE CLASS, ALTER WORK ACTION SET, ALTER THRESHOLD, or ALTER WORKLOAD statement and specify the COLLECT ACTIVITY DATA keywords.
6. Use the COMMIT statement to commit your changes.

Results

Note: Individual activity collection is more expensive than workload management statistics collection. You should try to set up activity collection to collect as few activities as possible. For example, if you need to investigate activities submitted by a specific application, you could isolate that application by creating a workload or service class specifically for that application, and only enable activity collection for that workload or service class.

You might not always know in advance that you will want to capture an activity. For example, you might have a query that is taking a long time to run and you want to collect information about it for later analysis. In this situation, it is too late to specify the COLLECT ACTIVITY DATA keyword on the DB2 workload management objects, because the activity has already entered the system. In this situation, you can use the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure. The WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure sends information about an executing activity to the active ACTIVITIES event monitor. You identify the activity to be collected using the application handle, unit of work identifier, and activity identifier. Information about the activity is immediately be sent to the ACTIVITIES event monitor when the procedure is invoked: you do not need to wait for the activity to complete.

Chapter 30. Explain facility

The DB2 explain facility provides detailed information about the access plan that the optimizer chooses for an SQL or XQuery statement.

The information provided describes the decision criteria that are used to choose the access plan. The information can also help you to tune the statement or your instance configuration to improve performance. More specifically, explain information can help you with the following tasks:

- Understanding how the database manager accesses tables and indexes to satisfy your query.
- Evaluating your performance-tuning actions. After altering a statement or making a configuration change, examine the new explain information to determine how your action has affected performance.

The captured information includes the following information:

- The sequence of operations that were used to process the query
- Cost information
- Predicates and selectivity estimates for each predicate
- Statistics for all objects that were referenced in the SQL or XQuery statement at the time that the explain information was captured
- Values for host variables, parameter markers, or special registers that were used to reoptimize the SQL or XQuery statement

The explain facility is invoked by issuing the EXPLAIN statement, which captures information about the access plan chosen for a specific explainable statement and writes this information to explain tables. You must create the explain tables prior to issuing the EXPLAIN statement. You can also set CURRENT EXPLAIN MODE or CURRENT EXPLAIN SNAPSHOT, special registers that control the behavior of the explain facility.

For privileges and authorities that are required to use the explain utility, see the description of the EXPLAIN statement. The EXPLAIN authority can be granted to an individual who requires access to explain information but not to the data that is stored in the database. This authority is a subset of the database administrator authority and has no inherent privilege to access data stored in tables.

To display explain information, you can use a command-line tool. The tool that you use determines how you set the special registers that control the behavior of the explain facility. If you expect to perform detailed analysis with one of the command-line utilities or with custom SQL or XQuery statements against the explain tables, capture all explain information.

In IBM Data Studio Version 3.1 or later, you can generate a diagram of the current access plan for an SQL or XPATH statement. For more details, see Diagramming access plans with Visual Explain.

Tuning SQL statements using the explain facility

The explain facility is used to display the query access plan that was chosen by the query optimizer to run an SQL statement.

It contains extensive details about the relational operations used to run the SQL statement, such as the plan operators, their arguments, order of execution, and costs. Because the query access plan is one of the most critical factors in query performance, it is important to understand explain facility output when diagnosing query performance problems.

Explain information is typically used to:
- Understand why application performance has changed
- Evaluate performance tuning efforts

Analyzing performance changes

To help you understand the reasons for changes in query performance, perform the following steps to obtain “before and after” explain information:
1. Capture explain information for the query before you make any changes, and save the resulting explain tables. Alternatively, you can save output from the **db2exfmt** utility. However, having explain information in the explain tables makes it easy to query them with SQL, and facilitates more sophisticated analysis. As well, it provides all of the obvious maintenance benefits of having data in a relational DBMS. The **db2exfmt** tool can be run at any time.
2. Save or print the current catalog statistics. You can also use the **db2look** command to help perform this task. In DB2 Version 9.7, you can collect an explain snapshot when the explain tables are populated. The explain snapshot contains all of the relevant statistics at the time that the statement is explained. The **db2exfmt** utility will automatically format the statistics that are contained in the snapshot. This is especially important when using automatic or real-time statistics collection, because the statistics used for query optimization might not yet be in the system catalog tables, or they might have changed between the time that the statement was explained and when the statistics were retrieved from the system catalog.
3. Save or print the data definition language (DDL) statements, including those for CREATE TABLE, CREATE VIEW, CREATE INDEX, and CREATE TABLESPACE. The **db2look** command will also perform this task.

The information that you collect in this way provides a reference point for future analysis. For dynamic SQL statements, you can collect this information when you run your application for the first time. For static SQL statements, you can also collect this information at bind time. It is especially important to collect this information before a major system change, such as the installation of a new service level or DB2 release, or before a significant configuration change, such as adding or dropping database partitions and redistributing data. This is because these types of system changes might result in an adverse change to access plans. Although access plan regression should be a rare occurrence, having this information available will help you to resolve performance regressions faster. To analyze a performance change, compare the information that you collected previously with information about the query and environment that you collect when you start your analysis.

As a simple example, your analysis might show that an index is no longer being used as part of an access plan. Using the catalog statistics information displayed by **db2exfmt**, you might notice that the number of index levels (NLEVELS column) is now substantially higher than when the query was first bound to the database. You might then choose to perform one of the following actions:
- Reorganize the index
- Collect new statistics for your table and indexes

- Collect explain information when rebinding your query

After you perform one of these actions, examine the access plan again. If the index is being used once again, query performance might no longer be a problem. If the index is still not being used, or if performance is still a problem, try a second action and examine the results. Repeat these steps until the problem is resolved.

Evaluating performance tuning efforts

You can take a number of actions to help improve query performance, such as adjusting configuration parameters, adding containers, or collecting fresh catalog statistics.

After you make a change in any of these areas, you can use the explain facility to determine the affect, if any, that the change has had on the chosen access plan. For example, if you add an index or materialized query table (MQT) based on index guidelines, the explain data can help you to determine whether the index or materialized query table is actually being used as expected.

Although the explain output provides information that allows you to determine the access plan that was chosen and its relative cost, the only way to accurately measure the performance improvement for a query is to use benchmark testing techniques.

Explain tables and the organization of explain information

An *explain instance* represents one invocation of the explain facility for one or more SQL or XQuery statements. The explain information that is captured in one explain instance includes information about the compilation environment and the access plan that is chosen to satisfy the SQL or XQuery statement that is being compiled.

For example, an explain instance might consist of any one of the following items:

- All eligible SQL or XQuery statements in one package, for static query statements. For SQL statements (including those that query XML data), you can capture explain information for CALL, compound SQL (dynamic), DELETE, INSERT, MERGE, REFRESH TABLE, SELECT, SELECT INTO, SET INTEGRITY, UPDATE, VALUES, and VALUES INTO statements. In the case of XQuery statements, you can obtain explain information for XQUERY db2-fn:xmlcolumn and XQUERY db2-fn:sqlquery statements.

  Note: REFRESH TABLE and SET INTEGRITY statements are compiled only dynamically.
- One particular SQL statement, for incremental bind SQL statements.
- One particular SQL statement, for dynamic SQL statements.
- Each EXPLAIN statement (dynamic or static).

The explain facility, which you can invoke by issuing the EXPLAIN statement or by using the section explain interfaces, captures information about the access plan that is chosen for a specific explainable statement and writes this information to explain tables. You must create the explain tables before issuing the EXPLAIN statement. To create the tables, use one of the following methods:

- Run the `EXPLAIN.DDL` script in the `misc` subdirectory of the `sqllib` subdirectory.
- Use the SYSPROC.SYSINSTALLOBJECTS procedure. You can also use this procedure to drop and validate explain tables.

You can create the tables under a specific schema and table space. You can find an example in the `EXPLAIN.DDL` file.

Explain tables can be common to more than one user. You can define tables for one user and then create aliases pointing to the defined tables for each additional user. Alternatively, you can define the explain tables under the SYSTOOLS schema. The explain facility uses the SYSTOOLS schema as the default if no explain tables or aliases are found under your session ID (for dynamic SQL or XQuery statements) or under the statement authorization ID (for static SQL or XQuery statements). Each user sharing common explain tables must hold the INSERT privilege on those tables.

The following table summarizes the purpose of each explain table.

Table 88. Summary of the explain tables

Table Name	Description
ADVISE_INDEX	Stores information about recommended indexes. The table can be populated by the query compiler or the **db2advis** command, or you can populate it. This table is used to get recommended indexes and to evaluate proposed indexes.
ADVISE_INSTANCE	Contains information about **db2advis** command execution, including start time. This table contains one row for each execution of the **db2advis** command.
ADVISE_MQT	Contains the following information: • The query that defines each recommended materialized query table (MQT) • The column statistics for each MQT, such as COLSTATS (in XML form) and NUMROWS • The sampling query to obtain detailed statistics for each MQT
ADVISE_PARTITION	Stores virtual database partitions that are generated and evaluated by the **db2advis** command.
ADVISE_TABLE	Stores the data definition language (DDL) statements for table creation, using the final Design Advisor recommendations for MQTs, multidimensional clustering tables (MDCs), and database partitioning.
ADVISE_WORKLOAD	Contains a row for each SQL or XQuery statement in a workload. The **db2advis** command uses this table to collect and store workload information.
EXPLAIN_ACTUALS	Contains the explain section actuals information.
EXPLAIN_ARGUMENT	Contains information about the unique characteristics of each operator, if any.
EXPLAIN_DIAGNOSTIC	Contains an entry for each diagnostic message that is produced for a particular instance of an explained statement in the EXPLAIN_STATEMENT table.
EXPLAIN_DIAGNOSTIC_DATA	Contains message tokens for diagnostic messages that are recorded in the EXPLAIN_DIAGNOSTIC table. The message tokens provide additional information that is specific to the execution of the SQL statement that generated the message.

Table 88. Summary of the explain tables (continued)

Table Name	Description
EXPLAIN_INSTANCE	Is the main control table for all explain information. Each row in the explain tables is linked to a unique row in this table. Basic information about the source of the SQL or XQuery statements being explained and environmental information are kept in this table.
EXPLAIN_OBJECT	Identifies the data objects that are required by the access plan that is generated to satisfy an SQL or XQuery statement.
EXPLAIN_OPERATOR	Contains all of the operators that the query compiler needs to satisfy an SQL or XQuery statement.
EXPLAIN_PREDICATE	Identifies the predicates that are applied by a specific operator.
EXPLAIN_STATEMENT	Contains the text of the SQL or XQuery statement for the different levels of explain information. The SQL or XQuery statement that you issued and the version that the optimizer uses to choose an access plan are stored in this table. When an explain snapshot is requested, additional explain information is recorded to describe the access plan that was selected by the query optimizer. This information is stored in the SNAPSHOT column of the EXPLAIN_STATEMENT table.
EXPLAIN_STREAM	Represents the input and output data streams between individual operators and data objects. The operators are represented in the EXPLAIN_OPERATOR table. The data objects are represented in the EXPLAIN_OBJECT table.
OBJECT_METRICS	Contains runtime statistics for each object that is referenced in a specific execution of a section at a specific time. If object statistics are collected on multiple members, this table contains a row for each member on which the object was referenced. If object statistics are collected for a partitioned object, this table contains a row for each data partition. This table contains information only if the activity event monitor captures section actuals.

Creating the explain tables

A number of steps are required to create explain tables.

About this task

To create explain snapshots, you must ensure that the following explain tables exist for your user ID:

- EXPLAIN_INSTANCE
- EXPLAIN_STATEMENT

To check if they exist, use the **LIST TABLES** command.

Procedure

If the explain tables do not exist, you must create them using the following instructions:

1. If the DB2 database manager has not already been started, issue the **db2start** command.
2. From the CLP prompt, connect to the database that you want to use.
3. Create the explain tables by using one of the following methods:
   - Call the SYSPROC.SYSINSTALLOBJECTS procedure:
     ```
     db2 CONNECT TO database-name
     db2 CALL SYSPROC.SYSINSTALLOBJECTS('EXPLAIN', 'C',
            CAST (NULL AS VARCHAR(128)), CAST (NULL AS VARCHAR(128)))
     ```
     This call creates the explain tables under the SYSTOOLS schema. To create them under a different schema, specify a schema name as the last parameter in the call.
   - Run the EXPLAIN.DDL command file:
     ```
     db2 CONNECT TO database-name
     db2 -tf EXPLAIN.DDL
     ```
     This command file creates explain tables under the current schema. It is at the DB2PATH\misc directory on Windows operating systems, and the INSTHOME/sqllib/misc directory on Linux and UNIX operating systems. DB2PATH is the location where you install your DB2 copy and INSTHOME is the instance home directory.

   Calling the SYSPROC.SYSINSTALLOBJECTS procedure is preferred over using the EXPLAIN.DDL file since it can automatically adapt to different database configurations. For example, if **BLOCKNONLOGGED** parameter is set to yes, then some statements in EXPLAIN.DDL fail because **NOT LOGGED** clause is used for LOB columns. However, if **BLOCKNONLOGGED** parameter is set to yes then the SYSPROC.SYSINSTALLOBJECTS procedure automatically avoids the use of **NOT LOGGED** clause.

Guidelines for capturing explain information

Explain data can be captured by request when an SQL or XQuery statement is compiled.

If incremental bind SQL or XQuery statements are compiled at run time, data is placed in the explain tables at run time, not at bind time. For these statements, the inserted explain table qualifier and authorization ID are that of the package owner, not of the user running the package.

Explain information is captured only when an SQL or XQuery statement is compiled. After initial compilation, dynamic query statements are recompiled when a change to the environment requires it, or when the explain facility is active. If you issue the same PREPARE statement for the same query statement, the query is compiled and explain data is captured every time that this statement is prepared or executed.

If a package is bound using the **REOPT ONCE** or **REOPT ALWAYS** bind option, SQL or XQuery statements containing host variables, parameter markers, global variables, or special registers are compiled, and the access path is created using real values for these variables if they are known, or default estimates if the values are not known at compilation time.

If the **REOPT ONCE** option is used, an attempt is made to match the specified SQL or XQuery statement with the same statement in the package cache. Values for this already re-optimized and cached query statement will be used to re-optimize the specified query statement. If the user has the required access privileges, the explain tables will contain the newly re-optimized access plan and the values that were used for re-optimization.

In a multi-partition database system, the statement should be explained on the same database partition on which it was originally compiled and re-optimized using **REOPT ONCE**; otherwise, an error is returned.

Capturing information in the explain tables

- Static or incremental bind SQL and XQuery statements

  Specify either **EXPLAIN ALL** or **EXPLAIN YES** options on the **BIND** or the **PREP** command, or include a static EXPLAIN statement in the source program.
- Dynamic SQL and XQuery statements

  Explain table information is captured in any of the following cases.
  - If the CURRENT EXPLAIN MODE special register is set to:
    - YES: The SQL and XQuery compiler captures explain data and executes the query statement.
    - EXPLAIN: The SQL and XQuery compiler captures explain data, but does not execute the query statement.
    - RECOMMEND INDEXES: The SQL and XQuery compiler captures explain data, and recommended indexes are placed in the ADVISE_INDEX table, but the query statement is not executed.
    - EVALUATE INDEXES: The SQL and XQuery compiler uses indexes that were placed by the user in the ADVISE_INDEX table for evaluation. In this mode, all dynamic statements are explained as though these virtual indexes were available. The query compiler then chooses to use the virtual indexes if they improve the performance of the statements. Otherwise, the indexes are ignored. To find out if proposed indexes are useful, review the EXPLAIN results.
    - REOPT: The query compiler captures explain data for static or dynamic SQL or XQuery statements during statement re-optimization at execution time, when actual values for host variables, parameter markers, global variables, or special registers are available.
- If the **EXPLAIN ALL** option has been specified on the **BIND** or **PREP** command, the query compiler captures explain data for dynamic SQL and XQuery statements at run time, even if the CURRENT EXPLAIN MODE special register is set to NO.

Capturing explain snapshot information

When an explain snapshot is requested, explain information is stored in the SNAPSHOT column of the EXPLAIN_STATEMENT table.

Explain snapshot data is captured when an SQL or XQuery statement is compiled and explain data has been requested, as follows:

- Static or incremental bind SQL and XQuery statements

An explain snapshot is captured when either the **EXPLSNAP ALL** or the **EXPLSNAP YES** clause is specified on the **BIND** or the **PREP** command, or when the source program includes a static EXPLAIN statement that uses a FOR SNAPSHOT or a WITH SNAPSHOT clause.

- Dynamic SQL and XQuery statements

  An explain snapshot is captured in any of the following cases.

  – You issue an EXPLAIN statement with a FOR SNAPSHOT or a WITH SNAPSHOT clause. With the former, only explain snapshot information is captured; with the latter, all explain information is captured.

  – If the CURRENT EXPLAIN SNAPSHOT special register is set to:

    - YES: The SQL and XQuery compiler captures explain snapshot data and executes the query statement.
    - EXPLAIN: The SQL and XQuery compiler captures explain snapshot data, but does not execute the query statement.

  – You specify the **EXPLSNAP ALL** option on the **BIND** or **PREP** command. The query compiler captures explain snapshot data at run time, even if the CURRENT EXPLAIN SNAPSHOT special register is set to NO.

Creating explain snapshots for dynamic SQL or XQuery statements

A number of steps are required to create explain snapshots for dynamic SQL or XQuery statements

Procedure

To create an *explain snapshot* for a dynamic SQL or XQuery statement:

1. If the database manager has not already been started, issue the **db2start** command.
2. Ensure that explain tables exist in your database.
   To do this, follow the instructions in “Creating the explain tables” on page 487.
3. From the CLP prompt, connect to the database that you want to use.
   For example, to connect to the SAMPLE database, issue the `connect to sample` command.
4. Create an explain snapshot for a dynamic SQL or XQuery statement, using either of the following commands from the CLP prompt:
   - To create an explain snapshot without executing the SQL or XQuery statement, issue the `set current explain snapshot=explain` command.
   - To create an explain snapshot and execute the SQL or XQuery statement, issue the `set current explain snapshot=yes` command.

   This command sets the explain special register. Once it is set, all subsequent SQL or XQuery statements are affected. For more information, see the CURRENT EXPLAIN SNAPSHOT special register and the SET CURRENT EXPLAIN SNAPSHOT statement.
5. Submit your SQL or XQuery statements from the CLP prompt.
6. Optional: To turn off the snapshot facility, issue the `set current explain snapshot=no` command after you submit your SQL or XQuery statements.

Creating explain snapshots for static SQL or XQuery statements

A number of steps are required create explain snapshots for static SQL or XQuery statements

Procedure

To create an *explain snapshot* for a static SQL or XQuery statement:

1. If the database manager has not already been started, issue the **db2start** command.
2. Ensure that explain tables exist in your database.
   To do this, follow the instructions in "Creating the explain tables" on page 487.
3. From the CLP prompt, connect to the database that you want to use.
   For example, to connect to the SAMPLE database, issue the `connect to sample` command.
4. Create an explain snapshot for a static SQL or XQuery statement by using the **EXPLSNAP** option when binding or preparing your application.
   For example, issue the `BIND` *`your_file`* `EXPLSNAP YES` command.

What to do next

For information about using the **EXPLSNAP** option see the CURRENT EXPLAIN SNAPSHOT special register, the **BIND** and **REBIND** commands, and the EXPLAIN statement.

Guidelines for capturing section explain information

The section explain functionality captures (either directly or via tooling) explain information about a statement using only the contents of the runtime section. The section explain is similar to the functionality provided by the **db2expln** command, but the section explain gives a level of detail approaching that which is provided by the explain facility.

By explaining a statement using the contents of the runtime section, you can obtain information and diagnostics about what will actually be run (or was run, if the section was captured after execution), as opposed to issuing an EXPLAIN statement which might produce a different access plan (for example, in the case of dynamic SQL, the statistics might have been updated since the last execution of the statement resulting in a different access plan being chosen when the EXPLAIN statement compiles the statement being explained).

The section explain interfaces will populate the explain tables with information that is similar to what is produced by an EXPLAIN statement. However, there are some differences. After the data has been written to the explain tables, it may be processed by any of the existing explain tools you want to use (for example, the **db2exfmt** command).

Section explain interfaces

There are four interface procedures, in the following list, that can perform a section explain. The procedures differ by only the input that is provided (that is, the means by which the section is located):

EXPLAIN_FROM_ACTIVITY
Takes application ID, activity ID, uow ID, and activity event monitor name

as input. The procedure searches for the section corresponding to this activity in the activity event monitor (an SQL activity is a specific execution of a section). A section explain using this interface contains section actuals because a specific execution of the section is being performed.

EXPLAIN_FROM_CATALOG
Takes package name, package schema, unique ID, and section number as input. The procedure searches the catalog tables for the specific section.

EXPLAIN_FROM_DATA
Takes executable ID, section, and statement text as input.

EXPLAIN_FROM_SECTION
Takes executable ID and location as input, where location is specified by using one of the following:

- In-memory package cache
- Package cache event monitor name

The procedure searches for the section in the given location.

An executable ID uniquely and consistently identifies a section. The executable ID is an opaque, binary token generated at the data server for each section that has been executed. The executable ID is used as input to query monitoring data for the section, and to perform a section explain.

In each case, the procedure performs an explain, using the information contained in the identified runtime section, and writes the explain information to the explain tables identified by an *explain_schema* input parameter. It is the responsibility of the caller to perform a commit after invoking the procedure.

Differences between section explain and EXPLAIN statement output

The results obtained after issuing a section explain are similar to those collected after running the EXPLAIN statement. There are slight differences which are described per affected explain table and by the implications, if any, to the output generated by the **db2exfmt** utility.

The stored procedure output parameters EXPLAIN_REQUESTER, EXPLAIN_TIME, SOURCE_NAME, SOURCE_SCHEMA, and SOURCE_VERSION comprise the key used to look up the information for the section in the explain tables. Use these parameters with any existing explain tools (for example, **db2exfmt**) to format the explain information retrieved from the section.

EXPLAIN_INSTANCE table

The following columns are set differently for the row generated by a section explain:

- EXPLAIN_OPTION is set to value S
- SNAPSHOT_TAKEN is always set to N
- REMARKS is always NULL

EXPLAIN_STATEMENT table

When a section explain has generated an explain output, the EXPLAIN_LEVEL column is set to value S. It is important to note that the EXPLAIN_LEVEL column is part of the primary key of the table and part of the foreign key of most other EXPLAIN tables; hence, this EXPLAIN_LEVEL value will also be present in those other tables.

In the EXPLAIN_STATEMENT table, the remaining column values that are usually associated with a row with EXPLAIN_LEVEL = `P`, are instead present when EXPLAIN_LEVEL = `S`, with the exception of SNAPSHOT. SNAPSHOT is always NULL when EXPLAIN_LEVEL is `S`.

If the original statement was not available at the time the section explain was generated (for example, if the statement text was not provided to the EXPLAIN_FROM_DATA procedure), STATEMENT_TEXT is set to the string `UNKNOWN` when EXPLAIN_LEVEL is set to `O`.

In the **db2exfmt** output for a section explain, the following extra line is shown after the optimized statement:

```
Explain level:    Explain from section
```

EXPLAIN_OPERATOR table

Considering all of the columns recording a cost, only the TOTAL_COST and FIRST_ROW_COST columns are populated with a value after a section explain. All the other columns recording cost have a value of `-1`.

In the **db2exfmt** output for a section explain, the following differences are obtained:

- In the access plan graph, the I/O cost is shown as `NA`
- In the details for each operator, the only costs shown are `Cumulative Total Cost` and `Cumulative First Row Cost`

EXPLAIN_PREDICATE table

No differences.

EXPLAIN_ARGUMENT table

A small number of argument types are not written to the EXPLAIN_ARGUMENT table when a section explain is issued.

EXPLAIN_STREAM table

The following columns do not have values after a section explain:

- SINGLE_NODE
- PARTITION_COLUMNS
- SEQUENCE_SIZES

The following column always has a value of `-1` after a section explain:

- PREDICATE_ID

The following columns will have values only for streams originating from a base table object or default to no value and -1 respectively after a section explain:

- COLUMN_NAMES
- COLUMN_COUNT

In the **db2exfmt** output for a section explain, the information from these listed columns is omitted from the `Input Streams` and `Output Streams` section for each operator when they do not have values, or have a value of -1.

EXPLAIN_OBJECT table

After issuing a section explain, the STATS_SRC column is always set to an empty string and the CREATE_TIME column is set to NULL.

The following columns always have values of `-1` after a section explain:
- COLUMN_COUNT
- WIDTH
- FIRSTKEYCARD
- FIRST2KEYCARD
- FIRST3KEYCARD
- FIRST4KEYCARD
- SEQUENTIAL_PAGES
- DENSITY
- AVERAGE_SEQUENCE_GAP
- AVERAGE_SEQUENCE_FETCH_GAP
- AVERAGE_SEQUENCE_PAGES
- AVERAGE_SEQUENCE_FETCH_PAGES
- AVERAGE_RANDOM_PAGES
- AVERAGE_RANDOM_FETCH_PAGES
- NUMRIDS
- NUMRIDS_DELETED
- NUM_EMPTY_LEAFS
- ACTIVE_BLOCKS
- NUM_DATA_PART

The following columns will also have values of `-1` after a section explain for partitioned objects:
- OVERHEAD
- TRANSFER_RATE
- PREFETCHSIZE

In the **db2exfmt** output for a section explain, the information from these listed columns is omitted from the per-table and per-index statistical information found near the bottom of the output.

Section explain does not include compiler-referenced objects in its output (that is, rows where OBJECT_TYPE starts with a +). These objects are not shown in the **db2exfmt** output.

Capturing and accessing section actuals

Section actuals are runtime statistics collected during the execution of the section for an access plan. To capture a section with actuals, you use the activity event monitor. To access the section actuals, you perform a section explain using the EXPLAIN_FROM_ACTIVITY stored procedure.

To be able to view section actuals, you must perform a section explain on a section for which section actuals were captured (that is, both the section and the section actuals are the inputs to the explain facility). Information about enabling, capturing, and accessing section actuals is provided here.

Enabling section actuals

Section actuals will only be updated at runtime if they have been enabled. Enable section actuals for the entire database using the **section_actuals** database configuration parameter or for a specific application using the WLM_SET_CONN_ENV procedure.

Section actuals will only be updated at runtime if they have been enabled. Enable section actuals using the **section_actuals** database configuration parameter. To enable section actuals, set the parameter to BASE (the default value is NONE). For example:

```
db2 update database configuration using section_actuals base
```

To enable section actuals for a specific application, use the WLM_SET_CONN_ENV procedure and specify BASE for the **section_actuals** element. For example:

```
CALL WLM_SET_CONN_ENV(NULL,
     '<collectactdata>WITH DETAILS, SECTION</collectactdata>
      <collectsectionactuals>BASE</collectsectionactuals>
     ')
```

Note:

1. The setting of the **section_actuals** database configuration parameter that was in effect at the start of the unit of work is applied to all statements in that unit of work. When the **section_actuals** database configuration parameter is changed dynamically, the new value will not be seen by an application until the next unit of work.
2. The **section_actuals** setting specified by the **WLM_SET_CONN_ENV** procedure for an application takes effect immediately. Section actuals will be collected for the next statement issued by the application.
3. Section actuals cannot be enabled if automatic statistics profile generation is enabled (SQLCODE -5153).

Capturing section actuals

The mechanism for capturing a section, with section actuals, is the activity event monitor. An activity event monitor writes out details of an activity when the activity completes execution, if collection of activity information is enabled. Activity information collection is enabled using the COLLECT ACTIVITY DATA clause on a workload, service class, threshold, or work action. To specify collection of a section and actuals (if the latter is enabled), the SECTION option of the COLLECT ACTIVITY DATA clause is used. For example, the following statement indicates that any SQL statement, issued by a connection associated with the WL1 workload, will have information (including section and actuals) collected by any active activity event monitor when the statement completes:

```
ALTER WORKLOAD WL1 COLLECT ACTIVITY DATA WITH DETAILS,SECTION
```

In a partitioned database environment, section actuals are captured by an activity event monitor on all partitions where the activity was executed, if the statement being executed has a COLLECT ACTIVITY DATA clause applied to it and the COLLECT ACTIVITY DATA clause specifies both the SECTION keyword and the

ON ALL DATABASE PARTITIONS clause. If the ON ALL DATABASE PARTITIONS clause is not specified, then actuals are captured on only the coordinator partition. In addition, besides the COLLECT ACTIVITY DATA clause on a workload, service class, threshold, or work action, activity collection can be enabled (for an individual application) using the WLM_SET_CONN_ENV procedure with a second argument that includes the collectactdata tag with a value of "WITH DETAILS, SECTION".

Limitations

The limitations, with respect to the capture of section actuals, are the following:

- Section actuals will not be captured when the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure is used to send information about a currently executing activity to an activity event monitor. Any activity event monitor record generated by the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure will have a value of 1 in its partial_record column.
- When a reactive threshold has been violated, section actuals will be captured on only the coordinator partition.
- Explain tables must be migrated to DB2 Version 9.7 Fix Pack 1, or later, before section actuals can be accessed using a section explain. If the explain tables have not been migrated, the section explain will work, but section actuals information will not be populated in the explain tables. In this case, an entry will be written to the EXPLAIN_DIAGNOSTIC table.
- Existing DB2 V9.7 activity event monitor tables (in particular, the activity table) must be recreated before section actuals data can be captured by the activity event monitor. If the activity logical group does not contain the SECTION_ACTUALS column, a section explain may still be performed using a section captured by the activity event monitor, but the explain will not contain any section actuals data.

Accessing section actuals

Section actuals can be accessed using the EXPLAIN_FROM_ACTIVITY procedure. When you perform a section explain on an activity for which section actuals were captured, the EXPLAIN_ACTUALS explain table will be populated with the actuals information.

Note: Section actuals are only available when a section explain is performed using the EXPLAIN_FROM_ACTIVITY procedure.

The EXPLAIN_ACTUALS table is the child table of the existing EXPLAIN_OPERATOR explain table. When EXPLAIN_FROM_ACTIVITY is invoked, if the section actuals are available, the EXPLAIN_ACTUALS table will be populated with the actuals data. If the section actuals are collected on multiple database partitions, there is one row per database partition for each operator in the EXPLAIN_ACTUALS table.

Analysis of section actuals information in explain output

Section actuals, when available, are displayed in different parts of the explain output. Where to find section actuals information, operator details, and object statistics in explain output is described here.

Section actuals in db2exfmt command graph output

If explain actuals are available, they are displayed in the graph under the estimated rows. Graph output includes actuals only for operators, not for objects. NA (not applicable) is displayed for objects in the graph.

An example of graph output from the**db2exfmt** command is as follows:

```
        Rows
     Rows Actual
       RETURN
       (   1)
        Cost
         I/O
         |
       3.21948  << The estimated rows that are used by the optimizer
         301    << The actuals rows that are collected in run time
        DTQ
       (   2)
       75.3961
         NA
         |
       3.21948
         130
       HSJOIN
       (   3)
       72.5927
         NA
      /--+---\
    674        260
    220        130
  TBSCAN     TBSCAN
  (   4)     (   5)
  40.7052    26.447
    NA         NA
    |          |
    337        130
    NA         NA  << Graph output does not include actuals for objects
 TABLE: FF  TABLE: FF
    T1         T2
```

In a partitioned database environment, the cardinality that is displayed in the graph is the average cardinality for the database partitions where the actuals are collected. The average is displayed because that is the value that is estimated by the optimizer. The actual average is a meaningful value to compare against the estimated average. In addition, a breakdown of section actuals per database partition is provided in the operator details output. You can examine these details to determine other information, such as total (across all partitions), minimum, and maximum.

Operator details in db2exfmt command output

The actual cardinality for an operator is displayed in the stream section following the line containing `Estimated number of rows` (`Actual number of rows` in the explain output). In a partitioned database environment, if the operator is running on more than one database member, the actual cardinality that is displayed is the average cardinality for the environment. The values per database partition are displayed under a separate section, `Explain Actuals`. This section is shown only for a partitioned database environment, but not in the serial mode. If the actuals are not available for a particular database partition, `NA` is displayed in the list of values per database partition next to the partition number. `Actual number of rows` in the section `Output Streams` is also shown as `NA`.

An example of operator details output from the **db2exfmt** command is as follows:

```
9) UNION : (Union)
   Cumulative Total Cost:     10.6858
   Cumulative First Row Cost:    9.6526

   Arguments:
   ---------
   UNIONALL: (UnionAll Parameterized Base Table)
      DISJOINT

   Input Streams:
   -------------
      5) From Operator #10

         Estimated number of rows:  30
         Actual number of rows:     63
         Partition Map ID:          3

      7) From Operator #11

         Estimated number of rows:  16
         Actual number of rows:     99
         Partition Map ID:          3


   Output Streams:
   --------------
      8) To Operator #8

         Estimated number of rows:  30
         Actual number of rows:     162
         Partition Map ID:          3

   Explain Actuals:      << This section is shown only show
                              in a partitioned database environment
   ----------------
      DB Partition number   Cardinality
      -------------------   -----------
               1               193
               2               131
```

Object statistics in db2exfmt command output

The explain output includes statistics for each object that is used in the access plan. For partitioned tables and indexes, the statistics are per data partition. In a partitioned database environment or DB2 pureScale environment, the statistics are per member. If the statistics are not available for a particular member, NA is displayed in the values list for that member next to the member number.

The following example shows how object statistics are displayed in the output of the **db2exfmt** command:

```
Runtime statistics for objects Used in Access Plan:
-----------------------------------------------------

        Schema: GOSALES
        Name:   ORDER_DETAILS
        Type:   Table

                        Member 0
                        ---------

                                Metrics
                                -----------------
                                lock_wait_time:85899
```

```
lock_wait_time_global:25769
lock_waits_local:21474
lock_waits_global:85899
lock_escals_local:17179
lock_escals_global:2
direct_writes:12884
direct_read_reqs:1
pool_data_gbp_invalid_pages:446
pool_data_lbp_pages_found:445
pool_xda_l_reads:446
pool_xda_p_reads:15
```

Guidelines for using explain information

You can use explain information to understand why application performance has changed or to evaluate performance tuning efforts.

Analysis of performance changes

To help you understand the reasons for changes in query performance, you need "before and after" explain information, which you can obtain by performing the following steps:

1. Capture explain information for the query before you make any changes and save the resulting explain tables. Alternatively, save output from the **db2exfmt** explain tool.
2. Save or print the current catalog statistics. You could use the **db2look** productivity tool to help you perform this task.
3. Save or print the data definition language (DDL) statements, including CREATE TABLE, CREATE VIEW, CREATE INDEX, or CREATE TABLESPACE.

The information that you collect in this way provides a reference point for future analysis. For dynamic SQL or XQuery statements, you can collect this information when you run your application for the first time. For static SQL and XQuery statements, you can collect this information at bind time. To analyze a performance change, compare the information that you collect with this reference information that was collected previously.

For example, your analysis might show that an index is no longer being used when determining an access path. Using the catalog statistics information, you might notice that the number of index levels (the NLEVELS column) is now substantially higher than when the query was first bound to the database. You might then choose to perform one of the following actions:

- Reorganize the index
- Collect new statistics for your table and indexes
- Collect explain information when rebinding your query

After you perform one of these actions, examine the access plan again. If the index is being used, query performance might no longer be a problem. If the index is still not being used, or if performance is still a problem, choose another action from this list and examine the results. Repeat these steps until the problem is resolved.

Evaluation of performance tuning efforts

You can take a number of actions to help improve query performance, such as updating configuration parameters, adding containers, collecting fresh catalog statistics, and so on.

After you make a change in any of these areas, use the explain facility to determine what affect, if any, the change has had on the chosen access plan. For example, if you add an index or materialized query table (MQT) based on the index guidelines, the explain data can help you to determine if the index or MQT is actually being used as expected.

Although the explain output enables you to determine the access plan that was chosen and its relative cost, the only way to accurately measure the performance improvement for a specific query is to use benchmark testing techniques.

Guidelines for analyzing explain information

The primary use for explain information is the analysis of access paths for query statements. There are a number of ways in which analyzing the explain data can help you to tune your queries and environment.

Consider the following kinds of analysis:

- Index use

  The proper indexes can significantly benefit performance. Using explain output, you can determine whether the indexes that you have created to help a specific set of queries are being used. Look for index usage in the following areas:
  - Join predicates
  - Local predicates
  - GROUP BY clause
  - ORDER BY clause
  - WHERE XMLEXISTS clause
  - The select list

  You can also use the explain facility to evaluate whether a different index or no index at all might be better. After you create a new index, use the **RUNSTATS** command to collect statistics for that index, and then recompile your query. Over time, you might notice (through explain data) that a table scan is being used instead of an index scan. This can result from a change in the clustering of the table data. If the index that was previously being used now has a low cluster ratio, you might want to:
  - Reorganize the table to cluster its data according to that index
  - Use the **RUNSTATS** command to collect statistics for both index and table
  - Recompile the query

  To determine whether reorganizing the table has improved the access plan, examine explain output for the recompiled query.
- Access type

  Analyze the explain output, and look for data access types that are not usually optimal for the type of application that you are running. For example:
  - Online transaction processing (OLTP) queries

    OLTP applications are prime candidates for index scans with range-delimiting predicates, because they tend to return only a few rows that are qualified by an equality predicate against a key column. If your OLTP queries are using a table scan, you might want to analyze the explain data to determine why an index scan is not being used.
  - Browse-only queries

    The search criteria for a “browse” type query can be very vague, resulting in a large number of qualifying rows. If users usually look at only a few screens of output data, you might specify that the entire answer set need not be computed before some results are returned. In this case, the goals of the user are different than the basic operating principle of the optimizer, which attempts to minimize resource consumption for the entire query, not just the first few screens of data.

For example, if the explain output shows that both merge scan join and sort operators were used in the access plan, the entire answer set will be materialized in a temporary table before any rows are returned to the application. In this case, you can attempt to change the access plan by using the OPTIMIZE FOR clause on the SELECT statement. If you specify this option, the optimizer can attempt to choose an access plan that does not produce the entire answer set in a temporary table before returning the first rows to the application.

- Join methods

  If a query joins two tables, check the type of join being used. Joins that involve more rows, such as those in decision-support queries, usually run faster with a hash join or a merge join. Joins that involve only a few rows, such as those in OLTP queries, typically run faster with nested-loop joins. However, there might be extenuating circumstances in either case-such as the use of local predicates or indexes-that could change how these typical joins work.

Tools for collecting and analyzing explain information

The DB2 database server has a comprehensive explain facility that provides detailed information about the access plan that the optimizer chooses for an SQL or XQuery statement.

The tables that store explain data are accessible on all supported platforms and contain information for both static and dynamic SQL and XQuery statements. Several tools are available to give you the flexibility that you need to capture, display, and analyze explain information.

Detailed query optimizer information that enables the in-depth analysis of an access plan is stored in explain tables that are separate from the actual access plan itself. Use one or more of the following methods to get information from the explain tables:

- Use the **db2exfmt** tool to display explain information in formatted output.
- Write your own queries against the explain tables. Writing your own queries enables the easy manipulation of output, comparisons among different queries, or comparisons among executions of the same query over time.

Use the **db2expln** tool to see the access plan information that is available for one or more packages of static SQL or XQuery statements. This utility shows the actual implementation of the chosen access plan; it does not show optimizer information. By examining the generated access plan, the **db2expln** tool provides a relatively compact, verbal overview of the operations that will occur at run time.

The command line explain tools can be found in the misc subdirectory of the sqllib directory.

The following table summarizes the different tools that are available with the DB2 explain facility. Use this table to select the tool that is most suitable for your environment and needs.

Table 89. Explain Facility Tools

Desired characteristics	Explain tables	db2expln	db2exfmt
Text output		Yes	Yes
"Quick and dirty" static SQL and XQuery analysis		Yes	
Static SQL and XQuery support	Yes	Yes	Yes
Dynamic SQL and XQuery support	Yes	Yes	Yes
CLI application support	Yes		Yes
Available to DRDA® Application Requesters	Yes		
Detailed optimizer information	Yes		Yes
Suited for analysis of multiple statements	Yes	Yes	Yes
Information is accessible from within an application	Yes		

In addition to these tools, you can use IBM Data Studio Version 3.1 or later to generate a diagram of the current access plan for SQL or XPATH statements. For more details, see Diagramming access plans with Visual Explain.

Displaying catalog statistics that are in effect at explain time

The explain facility captures the statistics that are in effect when a statement is explained. These statistics might be different than those that are stored in the system catalog, especially if real-time statistics gathering is enabled. If the explain tables are populated, but an explain snapshot was not created, only some statistics are recorded in the EXPLAIN_OBJECT table.

To capture all catalog statistics that are relevant to the statement being explained, create an explain snapshot at the same time that explain tables are being populated, then use the SYSPROC.EXPLAIN_FORMAT_STATS scalar function to format the catalog statistics in the snapshot.

If the **db2exfmt** tool is used to format the explain information, and an explain snapshot was collected, the tool automatically uses the SYSPROC.EXPLAIN_FORMAT_STATS function to display the catalog statistics.

SQL and XQuery explain tool

The **db2expln** command describes the access plan selected for SQL or XQuery statements.

You can use this tool to obtain a quick explanation of the chosen access plan when explain data was not captured. For static SQL and XQuery statements, **db2expln** examines the packages that are stored in the system catalog. For dynamic SQL and XQuery statements, **db2expln** examines the sections in the query cache.

The explain tool is located in the `bin` subdirectory of your instance `sqllib` directory. If **db2expln** is not in your current directory, it must be in a directory that appears in your PATH environment variable.

The **db2expln** command uses the `db2expln.bnd`, `db2exsrv.bnd`, and `db2exdyn.bnd` files to bind itself to a database the first time the database is accessed.

Description of db2expln output

Explain output from the **db2expln** command includes both package information and section information for each package.

- Package information includes the date of the bind operation and relevant bind options
- Section information includes the section number and the SQL or XQuery statement being explained

Explain output pertaining to the chosen access plan for the SQL or XQuery statement appears below the section information.

The steps of an access plan, or section, are presented in the order that the database manager executes them. Each major step is shown as a left-aligned heading with information about that step indented below it. Indentation bars are displayed in the left margin of the explain output for an access plan. These bars also mark the scope of each operation. Operations at a lower level of indentation, farther to the right, are processed before those that appear in the previous level of indentation.

The chosen access plan is based on an augmented version of the original SQL statement, the *effective SQL statement* if statement concentrator is enabled, or the XQuery statement that is shown in the output. Because the query rewrite component of the compiler might convert the SQL or XQuery statement into an equivalent but more efficient format, the access plan shown in explain output might differ substantially from what you expect. The explain facility, which includes the explain tables, and the SET CURRENT EXPLAIN MODE statement, shows the actual SQL or XQuery statement that was used for optimization in the form of an SQL- or XQuery-like statement that is created by reverse-translating the internal representation of the query.

When you compare output from **db2expln** to output from the explain facility, the operator ID option (-opids) can be useful. Each time that **db2expln** begins processing a new operator from the explain facility, the operator ID number is printed to the left of the explained plan. The operator IDs can be used to compare steps in the different representations of the access plan. Note that there is not always a one-to-one correspondence between the operators in explain facility output and the operations shown by **db2expln**.

Using access plans to self-diagnose performance problems with REFRESH TABLE and SET INTEGRITY statements

Invoking the explain utility against REFRESH TABLE or SET INTEGRITY statements enables you to generate access plans that can be used to self-diagnose performance problems with these statements. This can help you to better maintain your materialized query tables (MQTs).

To get the access plan for a REFRESH TABLE or a SET INTEGRITY statement, use either of the following methods:

- Use the EXPLAIN PLAN FOR REFRESH TABLE or EXPLAIN PLAN FOR SET INTEGRITY option on the EXPLAIN statement.
- Set the CURRENT EXPLAIN MODE special register to EXPLAIN before issuing the REFRESH TABLE or SET INTEGRITY statement, and then set the CURRENT EXPLAIN MODE special register to NO afterwards.

Restrictions

- The REFRESH TABLE and SET INTEGRITY statements do not qualify for re-optimization; therefore, the REOPT explain mode (or explain snapshot) is not applicable to these two statements.
- The WITH REOPT ONCE clause of the EXPLAIN statement, which indicates that the specified explainable statement is to be re-optimized, is not applicable to the REFRESH TABLE and SET INTEGRITY statements.

Scenario

This scenario shows how you can generate and use access plans from EXPLAIN and REFRESH TABLE statements to self-diagnose the cause of your performance problems.

1. Create and populate your tables. For example:
   ```
   create table t (
     i1 int not null,
     i2 int not null,
     primary key (i1)
   );

   insert into t values (1,1), (2,1), (3,2), (4,2);

   create table mqt as (
     select i2, count(*) as cnt from t group by i2
   )
   data initially deferred
   refresh deferred;
   ```
2. Issue the EXPLAIN and REFRESH TABLE statements, as follows:
   ```
   explain plan for refresh table mqt;
   ```

   This step can be replaced by setting the EXPLAIN mode on the SET CURRENT EXPLAIN MODE special register, as follows:
   ```
   set current explain mode explain;
   refresh table mqt;
   set current explain mode no;
   ```
3. Use the **db2exfmt** command to format the contents of the explain tables and obtain the access plan. This tool is located in the `misc` subdirectory of the instance `sqllib` directory.
   ```
   db2exfmt -d dbname -o refresh.exp -1
   ```
4. Analyze the access plan to determine the cause of the performance problem. In the previous example, if T is a large table, a table scan would be very expensive. Creating an index might improve the performance of the query.

Chapter 31. Problem-determination tools

Use the problem-determination tools and resources that are provided with your DB2 products to help you understand, isolate, and resolve problems.

DB2 diagnostic (db2diag) log files

The DB2 diagnostic **db2diag** log files are primarily intended for use by IBM Software Support for troubleshooting purposes. The administration notification log is primarily intended for troubleshooting use by database and system administrators. Administration notification log messages are also logged to the **db2diag** log files using a standardized message format.

Overview

With DB2 diagnostic and administration notification messages both logged within the **db2diag** log files, this often makes the **db2diag** log files the first location to examine in order to obtain information about the operation of your databases. Help with the interpretation of the contents of these diagnostic log files is provided in the topics listed in the "Related links" section. If your troubleshooting attempts are unable to resolve your problem and you feel you require assistance, you can contact IBM Software Support (for details, see the "Contacting IBM Software Support" topic). In gathering relevant diagnostic information that will be requested to be sent to IBM Software Support, you can expect to include your **db2diag** log files among other sources of information which includes other relevant logs, storage dumps, and traces.

The **db2diag** log file can exist in two different forms:

Single diagnostic log file
One active diagnostic log file, named db2diag.log, that grows in size indefinitely. This is the default form and it exists whenever the **diagsize** database manager configuration parameter has the value of 0 (the default value for this parameter is 0).

Rotating diagnostic log files
A single active log file (named db2diag.*N*.log, where *N* is the file name index that is a continuously growing number starting from 0), although a series of diagnostic log files can be found in the location defined by the **diagpath** configuration parameter, each growing until reaching a limited size, at which time the log file is closed and a new one is created and opened for logging with an incremented file name index (db2diag.*N+1*.log). It exists whenever the **diagsize** database manager configuration parameter has a nonzero value.

You can choose which of these two forms exist on your system by appropriately setting the **diagsize** database manager configuration parameter.

Configuration

The **db2diag** log files can be configured in size, location, and the types of diagnostic errors recorded by setting the following database manager configuration parameters:

diagsize

The value of **diagsize** decides what form of diagnostic log file will be adopted. If the value is 0, a single diagnostic log file will be adopted. If the value is not 0, rotating diagnostic log files will be adopted, and this nonzero value also specifies the total size of all rotating diagnostic log files and all rotating administration notification log files. The instance must be restarted for the new value of the **diagsize** parameter to take effect. See the "diagsize - Diagnostic log file size configuration parameter" topic for complete details.

diagpath

Diagnostic information can be specified to be written to **db2diag** log files in the location defined by the **diagpath** configuration parameter. See the "diagpath - Diagnostic data directory path configuration parameter" topic for complete details.

alt_diagpath

The **alt_diagpath** database manager configuration parameter provides an alternate diagnostic data directory path for storing diagnostic information. If the database manager fails to write to the path specified by **diagpath**, the path specified by **alt_diagpath** is used to store diagnostic information.

diaglevel

The types of diagnostic errors written to the **db2diag** log files can be specified with the **diaglevel** configuration parameter. See the "diaglevel - Diagnostic error capture level configuration parameter" topic for complete details.

Note: If the **diagsize** configuration parameter is set to a non-zero value, that value specifies the total size of the combination of all rotating administration notification log files and all rotating diagnostic log files contained within the diagnostic data directory. For example, if a system with 4 database partitions has **diagsize** set to 1 GB, the maximum total size of the combined notification and diagnostic logs can reach is 4 GB (4 x 1 GB).

Interpretation of diagnostic log file entries

Use the **db2diag** log files analysis tool (**db2diag**) to filter and format the **db2diag** log files. With the addition of administration notification log messages being logged to the **db2diag** log files using a standardized message format, viewing the **db2diag** log files first is a recommended choice to understand what has been happening to the database.

As an alternative to using **db2diag**, you can use a text editor to view the diagnostic log file on the machine where you suspect a problem to have occurred. The most recent events recorded are the furthest down the file.

Note: The administration notification (*instance_name*.nfy) and diagnostic (db2diag) logs grow *continuously* as single log files. When the **diagsize** database manager configuration parameter is set to a nonzero value, both the administration notification and the db2diag log files become a series of rotating log files (*instance_name.N*.nfy and db2diag.*N*.log) having a limited total size which is determined by the value of the **diagsize** configuration parameter.

The following example shows the header information for a sample log entry, with all the parts of the log identified.

Note: Not every log entry will contain all of these parts. Only the first several fields (timestamp to TID) and FUNCTION will be present in all the **db2diag** log file records.

```
2007-05-18-14.20.46.973000-240 1  I27204F655 2  LEVEL: Info 3
PID : 3228 4  TID : 8796 5  PROC : db2syscs.exe 6
INSTANCE: DB2MPP 7  NODE : 002 8  DB : WIN3DB1 9
APPHDL : 0-51 10  APPID: 9.26.54.62.45837.070518182042 11
AUTHID : UDBADM 12
EDUID : 8796 13  EDUNAME: db2agntp 14  (WIN3DB1) 2
FUNCTION: 15  DB2 UDB, data management, sqldInitDBCB, probe:4820
DATA #1 : 16  String, 26 bytes
Setting ADC Threshold to:
DATA #2 : unsigned integer, 8 bytes
1048576
```

Legend:

1. A timestamp and timezone for the message.

Note: Timestamps in the **db2diag** log files contain a time zone. For example: `2006-02-13-14.34.35.965000-300`, where "-300" is the difference between UTC (Coordinated Universal Time, formerly known as GMT) and local time at the application server in minutes. Thus -300 represents UTC - 5 hours, for example, EST (Eastern Standard Time).

2. The record ID field. The recordID of the **db2diag** log files specifies the file offset at which the current message is being logged (for example, “27204”) and the message length (for example, “655”) for the platform where the DB2 diagnostic log was created.

3. The diagnostic level of the message. The levels are `Info`, `Warning`, `Error`, `Severe`, `Critical`, and `Event`.

4. The process ID

5. The thread ID

6. The process name

7. The name of the instance generating the message.

8. For multi-partition systems, the database partition generating the message. (In a non-partitioned database, the value is "000".)

9. The database name

10. The application handle. This value aligns with that used in **db2pd** output and lock dump files. It consists of the coordinator partition number followed by the coordinator index number, separated by a dash.

11. Identification of the application for which the process is working. In this example, the process generating the message is working on behalf of an application with the ID 9.26.54.62.45837.070518182042.

A TCP/IP-generated application ID is composed of three sections

1. **IP address**: It is represented as a 32-bit number displayed as a maximum of 8 hexadecimal characters.
2. **Port number**: It is represented as 4 hexadecimal characters.
3. A **unique identifier** for the instance of this application.

Note: When the hexadecimal versions of the IP address or port number begin with 0 through to 9, they are changed to G through to P. For example, "0" is mapped to "G", "1" is mapped to "H", and so on. The IP address, AC10150C.NA04.006D07064947 is interpreted as follows: The IP address remains AC10150C, which translates to 172.16.21.12. The port number is NA04. The first character is "N", which maps to "7". Therefore, the hexadecimal form of the port number is 7A04, which translates to 31236 in decimal form.

This value is the same as the *appl_id* monitor element data. For detailed information about how to interpret this value, see the documentation for the *appl_id* monitor element.

To identify more about a particular application ID, either:

- Use the **LIST APPLICATIONS** command on a DB2 server or LIST DCS APPLICATIONS on a DB2 Connect gateway to view a list of application IDs. From this list, you can determine information about the client experiencing the error, such as its database partition name and its TCP/IP address.
- Use the **GET SNAPSHOT FOR APPLICATION** command to view a list of application IDs.
- Use the **db2pd -applications -db <dbname>** command.

12 The authorization identifier.

13 The engine dispatchable unit identifier.

14 The name of the engine dispatchable unit.

15. The product name ("DB2"), component name (“data management”), and function name (“sqlInitDBCB”) that is writing the message (as well as the probe point (“4820”) within the function).

16. The information returned by a called function. There may be multiple data fields returned.

Now that you have seen a sample **db2diag** log file entry, here is a list of all of the possible fields:

```
<timestamp><timezone>       <recordID>         LEVEL: <level> (<source>)
PID     : <pid>             TID  : <tid>       PROC : <procName>
INSTANCE: <instance>        NODE : <node>      DB   : <database>
APPHDL  : <appHandle>       APPID: <appID>
AUTHID  : <authID>
EDUID   : <eduID>           EDUNAME: <engine dispatchable unit name>
FUNCTION: <prodName>, <compName>, <funcName>, probe:<probeNum>
MESSAGE : <messageID>  <msgText>
CALLED  : <prodName>, <compName>, <funcName>   OSERR: <errorName> (<errno>)
RETCODE : <type>=<retCode> <errorDesc>
ARG #N  : <typeTitle>, <typeName>, <size> bytes
... argument ...
DATA #N : <typeTitle>, <typeName>, <size> bytes
... data ...
```

The fields which were not already explained in the example, are:

-

  `<source>` Indicates the origin of the logged error. (You can find it at the end of the first line in the sample.) The possible values are:
  - `origin` - message is logged by the function where error originated (inception point)
  - `OS` - error has been produced by the operating system
  - `received` - error has been received from another process (client/server)
  - `sent` - error has been sent to another process (client/server)

- MESSAGE Contains the message being logged. It consists of:
  - <messageID> - message number, for example, ECF=0x9000004A or DIA8604C
  - <msgText> - error description

  When the CALLED field is also present, <msgText> is an impact of the error returned by the CALLED function on the function logging a message (as specified in the FUNCTION field)
- CALLED This is the function that returned an error. It consists of:
  - <prodName> - The product name: "OS", "DB2", "DB2 Tools" or "DB2 Common"
  - <compName> - The component name ('-' in case of a system call)
  - <funcName> - The called function name
- OSERR This is the operating system error returned by the CALLED system call. (You can find it at the end of the same line as CALLED.) It consists of:
  - <errorName> - the system specific error name
  - <errno> - the operating system error number
- ARG This section lists the arguments of a function call that returned an error. It consists of:
  - <N> - The position of an argument in a call to the "called" function
  - <typeTitle> - The label associated with the Nth argument typename
  - <typeName> - The name of the type of argument being logged
  - <size> - The size of argument to be logged
- DATA This contains any extra data dumped by the logging function. It consists of:
  - <N> - The sequential number of data object being dumped
  - <typeTitle> - The label of data being dumped
  - <typeName> - The name of the type of data field being logged, for example, PD_TYPE_UINT32, PD_TYPE_STRING
  - <size> - The size of a data object

Interpreting the informational record of the db2diag log files

The first message in the **db2diag** log files should always be an informational record.

An example of an informational record is as follows:

```
2006-02-09-18.07.31.059000-300 I1H917            LEVEL: Event
PID     : 3140                TID  : 2864        PROC : db2start.exe
INSTANCE: DB2                 NODE : 000
FUNCTION: DB2 UDB, RAS/PD component, _pdlogInt, probe:120
START   : New Diagnostic Log file
DATA #1 : Build Level, 124 bytes
Instance "DB2" uses "32" bits and DB2 code release "SQL09010"
with level identifier "01010107".
Informational tokens are "DB2 v9.1.0.190", "s060121", "", Fix Pack "0".
DATA #2 : System Info, 1564 bytes
System: WIN32_NT MYSRVR Service Pack 2 5.1 x86 Family 15, model 2, stepping 4
CPU: total:1 online:1 Cores per socket:1 Threading degree per core:1
Physical Memory(MB): total:1024 free:617 available:617
Virtual  Memory(MB): total:2462 free:2830
Swap     Memory(MB): total:1438 free:2213
Information in this record is only valid at the time when this file was created
(see this record's time stamp)
```

The Informational record is output for **db2start** on every logical partition. This results in multiple informational records: one per logical partition. Since the informational record contains memory values which are different on every partition, this information might be useful.

Setting the error capture level of the diagnostic log files

The DB2 diagnostic (**db2diag**) log files are files that contain text information logged by DB2 database systems. This information is used for troubleshooting and much of it is primarily intended for IBM Software Support.

About this task

The types of diagnostic errors that are recorded in the **db2diag** log files are determined by the **diaglevel** database manager configuration parameter setting.

Procedure

- To check the current setting, issue the command **GET DBM CFG**.

  Look for the following variable:

  ```
  Diagnostic error capture level              (DIAGLEVEL) = 3
  ```

- To change the value dynamically, use the **UPDATE DBM CFG** command.

  To change a database manager configuration parameter online:

  ```
  db2 attach to instance-name
  db2 update dbm cfg using parameter-name value
  db2 detach
  ```

  For example:

  ```
  DB2 UPDATE DBM CFG USING DIAGLEVEL X
  ```

  where *X* is the notification level you want. If you are diagnosing a problem that can be reproduced, IBM Software Support personnel might suggest that you use **diaglevel** 4 while performing troubleshooting.

First occurrence data capture information

First occurrence data capture (FODC) collects diagnostic information about a DB2 instance, host or member when a problem occurs. FODC reduces the need to reproduce a problem to obtain diagnostic information, because diagnostic information can be collected as the problem occurs.

FODC can be invoked manually with the **db2fodc** command when you observe a problem or invoked automatically whenever a predetermined scenario or symptom is detected. After the diagnostic information has been collected, it is used to help determine the potential causes of the problem. In some cases, you might be able to determine the problem cause yourself, or involvement from IBM support personnel will be required.

Once execution of the **db2fodc** command has finished, the **db2support** tool must be executed to collect the resulting diagnostic files and prepare the FODC package to be submitted to IBM Support. The **db2support** command will collect the contents of all FODC package directories found or specified with the **-fodcpath** parameter. This is done to avoid additional requests, from IBM Support for diagnostic information.

Collecting diagnosis information based on common outage problems

Diagnostic information can be gathered automatically in a first occurrence data collection (FODC) package as the problem that affects an instance, host, or member is occurring. The information in the FODC package can also be collected manually.

Automatic collection of diagnostic information

The database manager invokes the **db2fodc** command for automatic First Occurrence Data Capture (FODC), which in turn invokes one of the DB2 call-out scripts (COS).

To correlate the outage with the DB2 diagnostic logs and the other troubleshooting files, a diagnostic message is written to both the administration notification and the **db2diag** log files. The FODC package directory name includes the FODC_ prefix, the outage type, the timestamp when the FODC directory was created, and the member or partition number where the problem occurred. The FODC package description file is placed in the new FODC package directory.

Table 90. Automatic FODC types and packages

Package	Description	Script executed
FODC_Trap_*timestamp_ memberNumber*	An instance wide trap has occurred	**db2cos_trap** (.bat)
FODC_Panic_*timestamp_ memberNumber*	Engine detected an incoherence and decided not to continue	**db2cos_trap** (.bat)
FODC_BadPage_*timestamp _memberNumber*	A Bad Page has been detected	**db2cos_datacorruption** (.bat)
FODC_DBMarkedBad_ *timestamp_ memberNumber*	A database has been marked bad due to an error	**db2cos** (.bat)
FODC_IndexError_ *timestamp_PID_EDUID _memberNumber*	An EDU wide index error occurred.	**db2cos_indexerror_short** (.bat) or **db2cos_indexerror_long** (.bat)
FODC_Member_*timestamp _memberNumber*	A member or partition has failed or has received a kill signal	**db2cos_member** (.bat)

Manual collection of diagnostic information

You use the **db2fodc** command manually when you suspect a problem is occurring. Problem scenarios that you can collect diagnostic data for include apparent system hangs, performance issues, or when an upgrade operation or instance creation did not complete as expected. When the **db2fodc** command is run manually, a new FODC package directory is created. The FODC package directory name includes the FODC_ prefix, the problem scenario, the timestamp when the FODC directory was created, and the member(s) or partition number(s) where FODC was performed.

Table 91. Manual FODC types and packages

Package	Description	Script executed
FODC_Clp_*timestamp_ member*	User invoked **db2fodc -clp** to collect environment and configuration related information, used to troubleshoot problems related to instance creation.	**db2cos_clp** script (.bat)
FODC_Connections_ *timestamp_member*	User invoked **db2fodc -connections** to collect connection-related diagnostic data, used to diagnose problems such as sudden spikes in the number of applications in the executing or compiling state or new database connections being denied.	**db2cos_threshold** script (.bat)
FODC_Cpu_*timestamp_ member*	User invoked **db2fodc -cpu** to collect processor-related performance and diagnostic data, used to diagnose problems such as high processor utilization rates, a high number of running processes, or high processor wait times.	**db2cos_threshold** script (.bat)
FODC_Hang_*timestamp_ memberList*	User invoked **db2fodc -hang** to collect data for hang troubleshooting (or severe performance)	**db2cos_hang** (.bat)
FODC_Memory_*timestamp _member*	User invoked **db2fodc -memory** to collect memory-related diagnostic data, used to diagnose problems such as no free memory available, swap space being used at a high rate, excessive paging or a suspected a memory leak.	**db2cos_threshold** script (.bat)
FODC_Perf_*timestamp_ memberList*	User invoked **db2fodc -perf** to collect data for performance troubleshooting	**db2cos_perf** (.bat)
FODC_Preupgrade_ *timestamp_member*	User invoked **db2fodc -preupgrade** to collect performance related information before a critical upgrade or update such as upgrading an instance or updating to the next fix pack	**db2cos_preupgrade** (.bat)

Table 91. Manual FODC types and packages (continued)

Package	Description	Script executed
Scripts located in FODC_IndexError_ ***timestamp_PID_EDUID_ memberList***	User could issue **db2fodc -indexerror** *FODC_IndexError_directory* [**basic | full**] (default is basic) to invoke the **db2dart** commands in the script(s). On partitioned database environments, use **db2_all** "<<+*node*#< **db2fodc -indexerror** *FODC_IndexError_directory* [**basic | full**]". The *node*# is the last number in the *FODC_IndexError_directory* directory name. An absolute path is required when using **db2fodc -indexerror** with the **db2_all** command.	**db2cos_indexerror_long** (.bat) or **db2cos_indexerror_short** (.bat)

First occurrence data capture configuration

First occurrence data capture configuration (FODC) behaviour, including the path used to store the FODC package, is controlled by the *DB2FODC* registry variable, which can be set persistently with the **db2set** command or changed dynamically (in-memory only) through the **db2pdcfg** command. FODC behavior can also be customized by updating the call-out scripts (COS) invoked during FODC.

Each partition or member in the instance has its own FODC settings, and you can control how FODC takes place at the partition or member level. If FODC settings exist both at the member or partition level and at the instance level, the member or partition level settings override the instance level settings. For manual FODC, settings can also be overridden by command line parameters you specify, such as the **-fodcpath** parameter. In partitioned or DB2 pureScale database environments, if you specify a list of members or partitions for manual FODC, the settings for the first member or partition specified are used.

Persistent settings made with the **db2set** command do not become effective until the instance is recycled; dynamic settings made with the **db2pdcfg** command are effective immediately and remain effective in memory until the instance is recycled.

To help you control how FODC packages are handled, several *DB2FODC* registry variable settings are available, but not all settings are available on all platforms. You can control the following behaviors through the *DB2FODC* registry variable:

- Where the generated FODC packages are stored (with the *FODCPATH* setting)
- Whether core dump files are generated or not (with the *DUMPCORE* setting)
- How big core dump files can become (with the *CORELIMIT* setting)
- Where the generated core files are stored (with the *DUMPDIR* setting)

FODC by default invokes a **db2cos** call-out script to collect diagnostic information when the database manager cannot continue processing due to a panic, trap, segmentation violation or exception. To help you control the call-out script that is invoked during FODC, several *COS* parameter settings are available. You can control the following behaviors through the *COS* parameter of the *DB2FODC* registry variable:

- Whether the **db2cos** script is invoked when the database manager cannot continue processing (with the *ON* and *OFF* setting; the default is *ON*)
- How often the **db2cos** script checks the size of the output files generated (with the *COS_SLEEP* setting)
- How long FODC should wait for the **db2cos** script to finish (with the *COS_TIMEOUT* setting)
- How often the **db2cos** script is invoked during a database manager trap (with the *COS_COUNT* setting)
- Whether the **db2cos** script is enabled when the SQLO_SIG_DUMP signal is received (with the *COS_SQLO_SIG_DUMP* setting)

FODC package directory settings (FODCPATH)

FODC packages can result in the generation of large volumes of diagnostic data that require space to store and can impose a significant processor usage on the system. You can control what directory path FODC sends diagnostic data to, so that you can pick a directory path with sufficient free space available.

The following order is used to determine what FODC path to use:

Automatic FODC

FODCPATH registry variable setting

The **FODCPATH** parameter for the **DB2FODC** registry variable can be set at the member or partition level and at the instance level. FODC uses the **FODCPATH** parameter setting for each partition or member, if set. If a partition or member level setting does not exist, the instance level setting is used.

No FODC path settings

By default, if you do not specify any **FODCPATH** setting at either the member or instance level, FODC sends diagnostic information to the current diagnostic directory path (**diagpath** or **alt_diagpath**).

Manual FODC

db2fodc -fodcpath command parameter option

When manually invoking the **db2fodc** command, you can indicate the location where the FODC package directory is created by specifying the **-fodcpath** parameter option together with the command. If you specify the **-fodcpath** parameter with a valid path name, the FODCpackage directory is created in that path.

FODCPATH registry variable setting

If you do not specify the **-fodcpath** parameter with the **db2fodc** command, and you specified a list of partitions or members, the **db2fodc** command uses the **FODCPATH** parameter setting for the **DB2FODC** registry variable of the first partition or member from the list specified. If the value for that **FODCPATH** parameter is not set, **db2fodc** uses the instance level **FODCPATH** setting. If you do not specify the **-fodcpath** parameter and do no specify a list of partitions or members, the **db2fodc** command first tries to use the **FODCPATH** parameter setting for the current partition or member; if not set, the instance level setting is used.

No FODC path settings

By default, if you do not specify any FODC path, first occurrence data capture sends diagnostic information to the current diagnostic directory path (**diagpath** or **alt_diagpath**).

Assume that you have a partitioned database environment with 3 members or partitions (0, 1, and 2). The following example shows how to set the FODC path persistently at the instance level for all 3 partitions or members using the **db2set** command:

```
db2set DB2FODC=FODCPATH=/home/hotel49/juntang/FODC
```

FODC path settings can also be performed persistently at the member level for each member, overriding the instance level setting. To make these settings effective, the instance must be recycled. For example, to change the FODC path on member 0, issue the following command:

```
db2set DB2FODC=FODCPATH=/home/hotel49/juntang/FODC/FODC0 -i juntang 0
```

If you now want to change the FODC path dynamically on member 1 and member 2, you use the following **db2pdcfg** commands. These settings are effective immediately and remain in memory until the instance is recycled.

```
db2pdcfg -fodc FODCPATH=/home/hotel49/juntang/FODC/FODC1 -member 1

db2pdcfg -fodc FODCPATH=/home/hotel49/juntang/FODC/FODC2 -member 2
```

If you want to know what the current FODC settings are for each member or partition in a system, you can use the **db2pdcfg -fodc -member all** command (in the example, output is abridged and only the FODC path output is shown):

```
Database Member 0
FODC package path (FODCPATH)= /home/hotel49/juntang/FODC/FODC0/

Database Member 1
FODC package path (FODCPATH)= /home/hotel49/juntang/FODC/FODC1/

Database Member 2
FODC package path (FODCPATH)= /home/hotel49/juntang/FODC/FODC2/
```

Customized data collection

The behavior of data collection by **db2fodc -hang** and **db2fodc -perf** is also controlled by parameters defined in the TOOL OPTIONS section of the DB2 call-out script that is invoked during FODC. These parameters can be customized by changing the script that is executed during FODC.

To customize the data collection on UNIX systems, copy the script placed in /bin/db2cos_*symptom* to /adm/db2cos_*symptom*, where *symptom* is either hang or perf. Once in this new directory, modify the script as you like. On Windows systems, simply modify the default script \bin\db2cos_*symptom*.bat. On UNIX systems, **db2fodc** first tries to execute the script in /adm/db2cos_*symptom*, and, if it is not found, executes the original script in /bin/db2cos_*symptom*. On Windows systems, the script \bin\db2cos_*symptom*.bat is always executed.

Data collected as part of FODC

First occurrence data capture (FODC) results in the creation of a FODC package directory and subdirectories where diagnostic information is collected. The parent package directory, subdirectories and files that get collected are collectively known as a FODC package.

Files containing diagnostic information that are collected by FODC

FODC collects diagnostic information from a number of sources. The exact diagnostic information captured by FODC depends on the type of problem encountered and might include:

Administration notification log (*instance_name*.nfy)

- Operating system: All
- Default location:
  - Linux and UNIX: Located in the directory specified by the **diagpath** database manager configuration parameter.
  - Windows: Use the Event Viewer Tool (**Start** > **Control Panel** > **Administrative Tools** > **Event Viewer**)
- Created automatically when the instance is created.
- When significant events occur, DB2 writes information to the administration notification log. The information is intended for use by database and system administrators. The type of message recorded in this file is determined by the **notifylevel** configuration parameter.

  Note: When the **diagsize** database manager configuration parameter is set to a nonzero value, the single administration notification log file behavior (*instance_name*.nfy) will be changed to a rotating log behavior (*instance_name.N*.nfy).

DB2 diagnostic log (db2diag log file)

- Operating system: All
- Default location: Located in the directory identified by the **diagpath** database manager configuration parameter.
- Created automatically when the instance is created.
- This text file contains diagnostic information about error and warnings encountered by the instance. This information is used for troubleshooting and is intended for technicians at IBM Software Support. The type of message recorded in this file is determined by the **diaglevel** database manager configuration parameter.

  Note: When the **diagsize** database manager configuration parameter is set to a nonzero value, the single diagnostic log file behavior (a single db2diag.log file) will be changed to a rotating log behavior (db2diag.*N*.log).

DB2 administration server (DAS) diagnostic log (db2dasdiag.log)

- Operating system: All
- Default location:
  - Linux and UNIX: Located in `DASHOME/das/dump`, where `DASHOME` is the home directory of the DAS owner
  - Windows: Located in "dump" folder, in the DAS home directory. For example: `C:\Program Files\IBM\SQLLIB\DB2DAS00\dump`
- Created automatically when the DAS is created.
- This text file contains diagnostic information about errors and warnings encountered by the DAS.

DB2 event log (db2eventlog.xxx, where xxx is the database partition number)

- Operating system: All

- Default location: Located in the directory specified by the **diagpath** database manager configuration parameter
- Created automatically when the instance is created.
- The DB2 event log file is a circular log of infrastructure-level events occurring in the database manager. The file is fixed in size, and acts as circular buffer for the specific events that are logged as the instance runs. Every time you stop the instance, the previous event log will be replaced, not appended. If the instance traps, a db2eventlog.XXX.crash file is also generated. These files are intended for use by IBM Software Support.

DB2 callout script (db2cos) output files

- Operating system: All
- Default location: Located in the directory specified by the **diagpath** database manager configuration parameter
- If db2cos scripts are executed as a consequence of an FODC outage, db2cos output files will be placed under the FODC directory that was created in the location specified by the **diagpath** database manager configuration parameter.
- Created automatically when a panic, trap or segmentation violation occurs. Can also be created during specific problem scenarios, as specified using the **db2pdcfg** command.
- The default db2cos script will invoke **db2pd** commands to collect information in an unlatched manner. The contents of the db2cos output files will vary depending on the commands contained in the db2cos script, such as operating system commands and other DB2 diagnosing tools. For more details on the tools that are executed with the db2cos script, open the script file in a text editor.
- The db2cos script is shipped under the `bin/` directory. On UNIX, this directory is read-only. To create your own modifiable version of this script, copy the db2cos script to the `adm/` directory. You are free to modify this version of the script. If the script is found in the `adm/` directory, it is that version that is run. Otherwise, the default version in the `bin/` directory is run.

Dump files

- Operating system: All
- Default location: Located in the directory specified by the **diagpath** database manager configuration parameter
- If these files are dumped during an FODC outage, they will be placed under the FODC directory.
- Created automatically when particular problem scenarios arise.
- For some error conditions, extra information is logged in binary files named after the failing process ID. These files are intended for use by IBM Software Support.

Trap files

- Operating system: All
- Default location: Located in the directory specified by the **diagpath** database manager configuration parameter
- If these files are dumped during an FODC outage, they will be placed under the FODC directory.

- Created automatically when the instance ends abnormally. Can also be created at will using the **db2pd** command.
- The database manager generates a trap file if it cannot continue processing due to a trap, segmentation violation, or exception.

Core files

- Operating system: Linux and UNIX
- Default location: Located in the directory specified by the **diagpath** database manager configuration parameter
- If these files are dumped during an FODC outage, they will be placed under the FODC directory.
- Created by the operating system when the DB2 instance terminates abnormally.
- Among other things, the core image will include most or all of the memory allocations of DB2, which may be required for problem descriptions.

FODC package path and contents

FODC creates the FODC package directory in the FODC path specified. You specify the FODC path through the **FODCPATH** registry variable setting or the **db2fodc -fodcpath** command parameter option. If you do not specify any FODC path, first occurrence data capture sends diagnostic information to the current diagnostic directory path (**diagpath** or **alt_diagpath**). A **db2diag** log file diagnostic message is logged to identify the directory name used for FODC. The capture of diagnostic information can generate a significant volume of diagnostic data, depending on what parameters are specified, and enough space must be available in the directory path where FODC stores diagnostic information. To avoid a scenario where FODC fills all the available space in the file system and impacts your data server, it is recommended that you specify a FODC path where FODC can store the diagnostic data.

For automatic FODC, a package is collected for the member or partition where the problem is occurring; if the problem is occurring on multiple members, multiple packages are collected in separate FODC package directories. The FODC package directory follows the naming convention FODC_*outageType_timestamp_member_number*, where *outageType* is the problem symptom, *timestamp* is the time of FODC invocation, and *member_number* is the member or partition number where the problem occurred. For example, when a trap occurs on member 1, FODC might automatically create a package named like `FODC_Trap_ 2010-11-17-20.58.30.695243_0001`.

For manual FODC, a package is collected for the member(s) or partition(s) you specify. The naming convention for the FODC package directory is FODC_*manualOutageType_timestamp_memberList*, where *manualOutageType* is the problem symptom, *timestamp* is the time of FODC invocation, and *memberList* is a list of the members or partitions where the problem occurred. For example, the manually issued command **db2fodc -hang -basic -member 1,2,3 -db sample** creates a manual FODC package for members 1,2 and 3, and might be named like `FODC_hang_ 2010-11-17-20.58.30.695243_0001.0002.0003`.

One or more of the following subdirectories is created under the FODC package directory:

- DB2CONFIG containing DB2 configuration output and files

- DB2PD containing **db2pd** output or output files
- DB2SNAPS containing DB2 snapshots
- DB2TRACE containing DB2 traces
- OSCONFIG containing operating system configuration files
- OSSNAPS containing operating system monitor information
- OSTRACE containing operating system traces

Not all of these directories might exist, depending on your FODC configuration and the outage type for which the **db2fodc** command is run.

FODC sends the following diagnostic information to the FODC package directory:

db2fodc -clp collects the following information:

- Operating system information
- Instance and database configuration information

db2fodc -hang collects the following information:

db2fodc -hang collects the following info:

- Basic operating system information. The problem could be due to OS level, patches, and so on.
- Basic DB2 configuration information.
- Operating system monitor information: vmstat, netstat, iostat, and so on.
  - 2 iterations at least: with timestamps saved
- Partial call stacks: DB2 stack traces of top CPU agents.
- Operating system trace: trace on AIX.
- Diagnostic information collected by **db2pd**.
- DB2 trace.
- Full DB2 call stacks.
- Second round of DB2 configuration information.
  - Including second DB2 trace collection.
- Snapshot information: **db2 get snapshot for** database, applications, tables, and so on.
  - Information will be collected per node in case of multiple logical nodes.

db2fodc -perf monitors the system possibly collecting the following information:

- Snapshots
- Stacktraces
- Virtual Memory (Vmstat)
- Input/Output information (Iostat)
- traces
- Some other information depending on the case. See the script for more details.

db2fodc -indexerror collects the following information:

- Basic Mode
  - `db2cos_indexerror_short(.bat)` script is run. See script for additional details.

    - If applicable **db2dart** commands exist in the script, the **db2dart /DD**, **db2dart /DI**, or both data formatting actions are run with number of pages limited to 100.
- Full Mode
    - db2cos_indexerror_short(.bat) and db2cos_indexerror_long(.bat) scripts are run. See scripts for additional details.
    - If applicable **db2dart** commands exist in the script db2cos_indexerror_short(.bat), the **db2dart /DD**, **db2dart /DI**, or both data formatting actions are run with number of pages limited to 100.
    - If applicable **db2dart** commands exist in the script db2cos_indexerror_long(.bat), the **db2dart /DD**, **db2dart /DI**, or both data formatting actions are run with no limit to the number of pages.
    - If applicable **db2dart** commands exist in the db2cos_indexerror_long(.bat) script, the **db2dart /T** command is run. This command requires the database be offline.

db2fodc -preupgrade collects the following information:

- Operating system information
- Instance and database configuration information, such as output of the **db2level** command, environment variables, output of the **db2 get dbm cfg** command, and the db2nodes.cfg file
- System catalog data and statistics, such as optimizer information collected by the **db2support -d** *dbname* **-c -s -cl 0** command
- Operating system monitoring data, such as output of the **netstat -v** and **ps -elf** commands
- System files
- Package information, as returned by the DB2 LIST PACKAGES FOR SCHEMA *schema-name* SHOW DETAIL command for all schema names
- Any FODC_Preupgrade directories found in db2dump/. These directories contain information such as performance data, top dynamic SQL queries, and explain plans
- The logfile from the **db2ckupgrade** command in /tmp/ db2ckupgrade.log.*processID*, if it exists
- Output from the **db2prereqcheck** command

The following diagnostic information is also included when you specify the members on which to collect:

- Snapshots (after turning on all monitor switches)
- The **db2pd** command output for the -everything, -agents, -applications, -mempools, and -fcm parameters
- The dynamic SQL statements used most frequently
- The query plans for SQL statements
- The explain plans for static packages

Manual **db2fodc** command invocation results in the creation of a log file named db2fodc_*symptom*.log in the FODC_*symptom* directory, where *symptom* is one of the collection types, such as hang or perf. Inside this file, the **db2fodc** command also stores status information and metadata describing the FODC package inside the FODC subdirectory. This file contains information about the type of FODC, the timestamp of the start and end of data collection, and other information useful for the analysis of the FODC package.

Automatic FODC data generation

When an outage occurs and automatic first occurrence data capture (FODC) is enabled, data is collected based on symptoms. The data collected is specific to what is needed to diagnose the outage.

One or many messages, including those defined as "critical" are used to mark the origin of an outage.

Trap files contain information such as:

- The amount of free virtual storage
- Values associated with the product's configuration parameters and registry variables at the time the trap occurred
- Estimated amount of memory used by the DB2 product at the time of the trap
- Information that provides a context for the outage

The raw stack dump might be included in an ASCII trap file.

Dump files that are specific to components within the database manager are stored in the appropriate FODC package directory.

Monitor and audit facilities using First Occurrence Data Capture (FODC)

If you find you are required to investigate monitor or audit facility problems, there are logs that contain information about the probable cause for the difficulties you may be experiencing.

DB2 audit log ("`db2audit.log`")

- Operating system: All
- Default location:
  - Windows: Located in the `$DB2PATH\`*`instance_name`*`\security` directory
  - Linux and UNIX: Located in the `$HOME\sqllib\security` directory, where $HOME is the instance owner's home directory
- Created when the **`db2audit`** facility is started.
- Contains audit records generated by the DB2 audit facility for a series of predefined database events.

DB2 governor log ("*`mylog.x`*", where *`x`* is the number of database partitions on which the governor is running)

- Operating system: All
- Default location:
  - Windows: Located in the `$DB2PATH\`*`instance_name`*`\log` directory
  - Linux and UNIX: Located in the `$HOME\sqllib\log` directory, where $HOME is the instance owner's home directory
- Created when using the governor utility. The base of the log file name is specified in the **`db2gov`** command.
- Records information about actions performed by the governor daemon (for example, forcing an application, reading the governor configuration file, starting or ending the utility) as well as errors and warnings.

Event monitor file (for example, "00000000.evt")

- Operating system: All

- Default location: When you create a file event monitor, all of the event records are written to the directory specified in the CREATE EVENT MONITOR statement.
- Generated by the event monitor when events occur.
- Contains event records that are associated with the event monitor.

db2ls command

The db2ls command lists where DB2 products are installed on your system and the DB2 product level. It can also list all or specific DB2 products and features for a particular installation path.

Listing DB2 database products installed on your system (Linux and UNIX)

On Linux and UNIX operating systems, use the db2ls command to list the DB2 database products and DB2 features installed on your system.

Before you begin

At least one DB2 Version 9 (or later) database product must already be installed by a root user for a symbolic link to the **db2ls** command to be available in the /usr/local/bin directory.

About this task

With the ability to install multiple copies of DB2 database products on your system and the flexibility to install DB2 database products and features in the path of your choice, you need a tool to help you keep track of what is installed and where it is installed. On supported Linux and UNIX operating systems, the **db2ls** command lists the DB2 products and features installed on your system, including the DB2 HTML documentation.

The **db2ls** command can be found both in the installation media and in a DB2 install copy on the system. The **db2ls** command can be run from either location. The **db2ls** command can be run from the installation media for all products except IBM Data Server Driver Package.

The **db2ls** command can be used to list:

- Where DB2 database products are installed on your system and list the DB2 database product level
- All or specific DB2 database products and features in a particular installation path

Restrictions

The output that the **db2ls** command lists is different depending on the ID used:

- When the **db2ls** command is run with root authority, only root DB2 installations are queried.
- When the **db2ls** command is run with a non-root ID, root DB2 installations and the non-root installation owned by matching non-root ID are queried. DB2 installations owned by other non-root IDs are not queried.

The **db2ls** command is the only method to query a DB2 database product. You *cannot* query DB2 database products using Linux or UNIX operating system native utilities, such as **pkginfo**, **rpm**, **SMIT**, or **swlist**. Any existing scripts containing a native installation utility that you use to query and interface with DB2 installations must change.

You *cannot* use the **db2ls** command on Windows operating systems.

Procedure

- To list the path where DB2 database products are installed on your system and list the DB2 database product level, enter:

  ```
  db2ls
  ```

  The command lists the following information for each DB2 database product installed on your system:
  - Installation path
  - Level
  - Fix pack
  - Special Install Number. This column is used by IBM DB2 Support.
  - Installation date. This column shows when the DB2 database product was last modified.
  - Installer UID. This column shows the UID with which the DB2 database product was installed.
- To list information about DB2 database products or features in a particular installation path the **q** parameter must be specified:

  ```
  db2ls -q -p -b baseInstallDirectory
  ```

  where:
  - **q** specifies that you are querying a product or feature. This parameter is mandatory.
  - **p** specifies that the listing displays products rather than listing the features.
  - **b** specifies the installation directory of the product or feature. This parameter is mandatory if you are not running the command from the installation directory.

Results

Depending on the parameters provided, the command lists the following information:

- Installation path. This is specified only once, not for each feature.
- The following information is displayed:
  - Response file ID for the installed feature, or if the **p** option is specified, the response file ID for the installed product. For example, `ENTERPRISE_SERVER_EDITION`.
  - Feature name, or if the **p** option is specified, product name.
  - Product version, release, modification level, fix pack level (VRMF). For example, `10.1.0.0`
  - Fix pack, if applicable. For example, if Fix Pack 1 is installed, the value displayed is 1. This includes interim fix packs, such as Fix Pack 1a.
- If any of the product's VRMF information do not match, a warning message displays at the end of the output listing. The message suggests the fix pack to apply.

db2mtrk command

You can use the db2mtrk command to generate a complete report of memory status, for instances, databases, agents, and applications. The command output includes memory pool allocation information such as current size, maximum size, and type of function for which memory is used.

Buffer pools memory allocation

You can use the db2mtrk command to view the amount of database memory that is allocated to buffer pools. The buffer pool heaps are always fully allocated so the memory tracker reports the same values for the current and maximum sizes of these heaps. If a bufferpool size is set to automatic, then the current and maximum size of the buffer pool heap will be adjusted over time based on workload and available memory.

Example 1

To report the database and instance memory usage every 10 seconds, issue the following command: *db2mtrk -v -i -d -r 10*.

Example 2

In addition to database and instance memory usage, to report detailed application memory usage that is grouped by application ID, issue the following command: *db2mtrk -a -v -i -d*.

db2pd command

You can use the db2pd command for monitoring and troubleshooting because it can return quick and immediate information from the DB2 memory sets.

Overview

The tool collects information without acquiring any latches or using any engine resources. It is therefore possible (and expected) to retrieve information that is changing while **db2pd** is collecting information; hence the data might not be completely accurate. If changing memory pointers are encountered, a signal handler is used to prevent **db2pd** from ending abnormally. This can result in messages such as "Changing data structure forced command termination" to appear in the output. Nonetheless, the tool can be helpful for troubleshooting. Two benefits to collecting information without latching include faster retrieval and no competition for engine resources.

If you want to capture information about the database management system when a specific SQLCODE, ZRC code or ECF code occurs, this can be accomplished using the **db2pdcfg -catch** command. When the errors are caught, the db2cos (callout script) is launched. The db2cos script can be dynamically altered to run any **db2pd** command, operating system command, or any other command needed to resolve the problems. The template db2cos script file is located in `sqllib/bin` on UNIX and Linux. On the Windows operating system, db2cos is located in the `$DB2PATH\bin` directory.

When adding a new node, you can monitor the progress of the operation on the database partition server, that is adding the node, using the **db2pd -addnode** command with the optional `oldviewapps` and `detail` parameters for more detailed information.

If you require a list of event monitors that are currently active or have been, for some reason, deactivated, run the **db2pd -gfw** command. This command also returns statistics and information about the targets, into which event monitors write data, for each fast writer EDU.

Examples

The following list is a collection of examples in which the **db2pd** command can be used to expedite troubleshooting:

- Example 1: Diagnosing a lockwait
- Example 2: Using the **-wlocks** parameter to capture all the locks being waited on
- Example 3: Using the **-apinfo** parameter to capture detailed runtime information about the lock owner and the lock waiter
- Example 4: Using the callout scripts when considering a locking problem
- Example 5: Mapping an application to a dynamic SQL statement
- Example 6: Monitoring memory usage
- Example 7: Determine which application is using up your table space
- Example 8: Monitoring recovery
- Example 9: Determining the amount of resources a transaction is using
- Example 10: Monitoring log usage
- Example 11: Viewing the sysplex list
- Example 12: Generating stack traces
- Example 13: Viewing memory statistics for a database partition
- Example 14: Monitoring the progress of index reorganization
- Example 15: Displaying the top EDUs by processor time consumption and displaying EDU stack information
- Example 16: Displaying agent event metrics

The results text show in the examples is an extract of the the **db2cmd** command ouput for better readability.

Example 1: Diagnosing a lockwait

If you run **db2pd -db** *databasename* **-locks -transactions -applications -dynamic**, the results are similar to the following ones:

```
Locks:
TranHdl Lockname                   Type Mode Sts Owner Dur HldCnt Att    ReleaseFlg
3       00020002000000040000000052 Row  ..X  G   3     1   0      0x0000 0x40000000
2       00020002000000040000000052 Row  ..X  W*  2     1   0      0x0000 0x40000000
```

For the database that you specified using the -db database name option, the first results show the locks for that database. The results show that TranHdl 2 is waiting on a lock held by TranHdl 3.

```
Transactions:
AppHandl [nod-index] TranHdl Locks State Tflag      Tflag2     ...
11       [000-00011] 2       4     READ  0x00000000 0x00000000 ...
12       [000-00012] 3       4     WRITE 0x00000000 0x00000000 ...
```

We can see that TranHdl 2 is associated with AppHandl 11 and TranHdl 3 is associated with AppHandl 12.

```
Applications:
AppHandl NumAgents CoorPid Status        C-AnchID C-StmtUID L-AnchID L-StmtUID Appid

12       1         1073336 UOW-Waiting   0        0         17       1         ...5602
11       1         1040570 UOW-Executing 17       1         94       1         ...5601
```

We can see that AppHandl 12 last ran dynamic statement 17, 1. AppHandl 11 is currently running dynamic statement 17, 1 and last ran statement 94, 1.

```
Dynamic SQL Statements:
AnchID StmtUID NumEnv NumVar NumRef NumExe Text
 17     1       1      1      2      2      update pdtest set c1 = 5
 94     1       1      1      2      2      set lock mode to wait 1
```

We can see that the text column shows the SQL statements that are associated with the lock timeout.

Example 2: Using the **-wlocks** parameter to capture all the locks being waited on

If you run **db2pd -wlocks -db pdtest**, results similar to the following ones are generated. They show that the first application (AppHandl 47) is performing an insert on a table and that the second application (AppHandl 46) is performing a select on that table:

```
venus@boson:/home/venus =>db2pd -wlocks -db pdtest

Database Partition 0 -- Database PDTEST -- Active -- Up 0 days 00:01:22

Locks being waited on :
AppHandl TranHdl Lockname                   Type Mode Conv Sts CoorEDU AppName AuthID AppID
47       8       00020004000000000840000652 Row  ..X       G   5160    db2bp   VENUS  ...13730
46       2       00020004000000000840000652 Row  .NS       W   5913    db2bp   VENUS  ...13658
```

Example 3: Using the **-apinfo** parameter to capture detailed runtime information about the lock owner and the lock waiter

The following sample output was generated under the same conditions as those for Example 2:

```
venus@boson:/home/venus =>db2pd -apinfo 47 -db pdtest

Database Partition 0 -- Database PDTEST -- Active -- Up 0 days 00:01:30

Application :
  Address :                0x0780000001676480
  AppHandl [nod-index] :   47       [000-00047]
  Application PID :        876558
  Application Node Name :  boson
  IP Address:              n/a
  Connection Start Time :  (1197063450)Fri Dec  7 16:37:30 2007
  Client User ID :         venus
  System Auth ID :         VENUS
  Coordinator EDU ID :     5160
  Coordinator Partition :  0
  Number of Agents :       1
  Locks timeout value :    4294967294 seconds
  Locks Escalation :       No
  Workload ID :            1
  Workload Occurrence ID : 2
  Trusted Context :        n/a
  Connection Trust Type :  non trusted
  Role Inherited :         n/a
  Application Status :     UOW-Waiting
  Application Name :       db2bp
  Application ID :         *LOCAL.venus.071207213730

  ClientUserID :           n/a
  ClientWrkstnName :       n/a
  ClientApplName :         n/a
  ClientAccntng :          n/a

  List of inactive statements of current UOW :
    UOW-ID :          2
    Activity ID :     1
    Package Schema :  NULLID
    Package Name :    SQLC2G13
    Package Version :
    Section Number :  203
    SQL Type :        Dynamic
    Isolation :       CS
    Statement Type :  DML, Insert/Update/Delete
    Statement :       insert into pdtest values 99


venus@boson:/home/venus =>db2pd -apinfo 46 -db pdtest

Database Partition 0 -- Database PDTEST -- Active -- Up 0 days 00:01:39

Application :
  Address :                0x0780000000D77A60
  AppHandl [nod-index] :   46       [000-00046]
  Application PID :        881102
  Application Node Name :  boson
  IP Address:              n/a
  Connection Start Time :  (1197063418)Fri Dec  7 16:36:58 2007
  Client User ID :         venus
  System Auth ID :         VENUS
  Coordinator EDU ID :     5913
  Coordinator Partition :  0
  Number of Agents :       1
  Locks timeou t value :    4294967294 seconds
  Locks Escalation :       No
  Workload ID :            1
  Workload Occurrence ID : 1
  Trusted Context :        n/a
  Connection Trust Type :  non trusted
  Role Inherited :         n/a
  Application Status :     Lock-wait
  Application Name :       db2bp
  Application ID :         *LOCAL.venus.071207213658

  ClientUserID :           n/a
  ClientWrkstnName :       n/a
  ClientApplName :         n/a
  ClientAccntng :          n/a

  List of active statements :
   *UOW-ID :          3
    Activity ID :     1
    Package Schema :  NULLID
    Package Name :    SQLC2G13
    Package Version :
    Section Number :  201
    SQL Type :        Dynamic
    Isolation :       CS
    Statement Type :  DML, Select (blockable)
    Statement :       select * from pdtest
```

Example 4: Using the callout scripts when considering a locking problem

To use the callout scripts, find the db2cos output files. The location of the files is controlled by the database manager configuration parameter **diagpath**. The contents of the output files will differ depending on what commands you enter in the db2cos script file. An example of the output provided when the db2cos script file contains a **db2pd -db sample -locks** command is as follows:

```
Lock Timeout Caught
Thu Feb 17 01:40:04 EST 2006
Instance DB2
Database: SAMPLE
Partition Number: 0
PID: 940
TID: 2136
Function: sqlplnfd
Component: lock manager
Probe: 999
Timestamp: 2006-02-17-01.40.04.106000
AppID: *LOCAL.DB2...
AppHdl:
...
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:06:53
Locks:
Address    TranHdl Lockname                   Type Mode Sts Owner Dur HldCnt Att Rlse
0x402C6B30 3       00020003000000040000000052 Row  ..X  W*  3     1   0       0  0x40
```

In the output, W* indicates the lock that experienced the timeout. In this case, a lockwait has occurred. A lock timeout can also occur when a lock is being converted to a higher mode. This is indicated by C* in the output.

You can map the results to a transaction, an application, an agent, or even an SQL statement with the output provided by other **db2pd** commands in the db2cos file. You can narrow down the output or use other commands to collect the information that you need. For example, you can use the **db2pd -locks wait** parameters to print only locks with a wait status. You can also use the **-app** and **-agent** parameters.

Example 5: Mapping an application to a dynamic SQL statement

The command **db2pd -applications -dynamic** reports the current and last anchor ID and statement unique ID for dynamic SQL statements. This allows direct mapping from an application to a dynamic SQL statement.

```
Applications:
Address            AppHandl [nod-index] NumAgents  CoorPid  Status
0x00000002006D2120 780      [000-00780] 1          10615    UOW-Executing

C-AnchID C-StmtUID  L-AnchID L-StmtUID  Appid
163      1          110      1          *LOCAL.burford.050202200412

Dynamic SQL Statements:
Address            AnchID StmtUID NumEnv NumVar NumRef NumExe Text
0x0000000220A02760 163    1       2      2      2      1      CREATE VIEW MYVIEW
0x0000000220A0B460 110    1       2      2      2      1      CREATE VIEW YOURVIEW
```

Example 6: Monitoring memory usage

The **db2pd -memblock** command can be useful when you are trying to understand memory usage, as shown in the following sample output:

```
All memory blocks in DBMS set.

Address            PoolID   PoolName   BlockAge   Size(Bytes) I LOC   File
0x0780000000740068 62       resynch    2          112         1 1746  1583816485
0x0780000000725688 62       resynch    1          108864      1 127   1599127346
0x07800000001F4348 57       ostrack    6          5160048     1 3047  698130716
0x07800000001B5608 57       ostrack    5          240048      1 3034  698130716
0x07800000001A0068 57       ostrack    1          80          1 2970  698130716
0x07800000001A00E8 57       ostrack    2          240         1 2983  698130716
```

```
0x07800000001A0208 57       ostrack    3          80          1 2999  698130716
0x07800000001A0288 57       ostrack    4          80          1 3009  698130716
0x0780000000700068 70       apmh       1          360         1 1024  3878879032
0x07800000007001E8 70       apmh       2          48          1 914   1937674139
0x0780000000700248 70       apmh       3          32          1 1000  1937674139
...
```

This is followed by the sorted 'per-pool' output:

```
Memory blocks sorted by size for ostrack pool:
PoolID     PoolName   TotalSize(Bytes)     TotalCount LOC   File
57         ostrack    5160048              1          3047  698130716
57         ostrack    240048               1          3034  698130716
57         ostrack    240                  1          2983  698130716
57         ostrack    80                   1          2999  698130716
57         ostrack    80                   1          2970  698130716
57         ostrack    80                   1          3009  698130716
Total size for ostrack pool: 5400576 bytes

Memory blocks sorted by size for apmh pool:
PoolID     PoolName   TotalSize(Bytes)     TotalCount LOC   File
70         apmh       40200                2          121   2986298236
70         apmh       10016                1          308   1586829889
70         apmh       6096                 2          4014  1312473490
70         apmh       2516                 1          294   1586829889
70         apmh       496                  1          2192  1953793439
70         apmh       360                  1          1024  3878879032
70         apmh       176                  1          1608  1953793439
70         apmh       152                  1          2623  1583816485
70         apmh       48                   1          914   1937674139
70         apmh       32                   1          1000  1937674139
Total size for apmh pool: 60092 bytes
...
```

The final section of output sorts the consumers of memory for the entire memory set:

```
All memory consumers in DBMS memory set:
PoolID     PoolName   TotalSize(Bytes)     %Bytes TotalCount %Count LOC   File
57         ostrack    5160048              71.90  1          0.07   3047  698130716
50         sqlch      778496               10.85  1          0.07   202   2576467555
50         sqlch      271784               3.79   1          0.07   260   2576467555
57         ostrack    240048               3.34   1          0.07   3034  698130716
50         sqlch      144464               2.01   1          0.07   217   2576467555
62         resynch    108864               1.52   1          0.07   127   1599127346
72         eduah      108048               1.51   1          0.07   174   4210081592
69         krcbh      73640                1.03   5          0.36   547   4210081592
50         sqlch      43752                0.61   1          0.07   274   2576467555
70         apmh       40200                0.56   2          0.14   121   2986298236
69         krcbh      32992                0.46   1          0.07   838   698130716
50         sqlch      31000                0.43   31         2.20   633   3966224537
50         sqlch      25456                0.35   31         2.20   930   3966224537
52         kerh       15376                0.21   1          0.07   157   1193352763
50         sqlch      14697                0.20   1          0.07   345   2576467555
...
```

You can also report memory blocks for private memory on UNIX and Linux operating systems. For example, if you run **db2pd -memb pid=159770**, results similar to the following ones are generated:

```
All memory blocks in Private set.

PoolID     PoolName     BlockAge   Size(Bytes) I LOC   File
88         private      1          2488        1 172   4283993058
88         private      2          1608        1 172   4283993058
88         private      3          4928        1 172   4283993058
88         private      4          7336        1 172   4283993058
88         private      5          32          1 172   4283993058
```

```
88         private      6          6728       1 172   4283993058
88         private      7          168        1 172   4283993058
88         private      8          24         1 172   4283993058
88         private      9          408        1 172   4283993058
88         private      10         1072       1 172   4283993058
88         private      11         3464       1 172   4283993058
88         private      12         80         1 172   4283993058
88         private      13         480        1 1534  862348285
88         private      14         480        1 1939  862348285
88         private      80         65551      1 1779  4231792244
Total set size: 94847 bytes

Memory blocks sorted by size:
PoolID     PoolName   TotalSize(Bytes)     TotalCount LOC   File
88         private    65551                1          1779  4231792244
88         private    28336                12         172   4283993058
88         private    480                  1          1939  862348285
88         private    480                  1          1534  862348285
Total set size: 94847 bytes
```

Example 7: Determine which application is using up your table space

Using **db2pd -tcbstats** command, you can identify the number of inserts for a table. The following example shows sample information for a user-defined global temporary table called TEMP1:

```
TCB Table Information:
TbspaceID TableID PartID ... TableName SchemaNm ObjClass DataSize LfSize LobSize XMLSize
3         2       n/a    ... TEMP1     SESSION  Temp     966      0      0       0

TCB Table Stats:
TableName Scans UDI PgReorgs ... Reads FscrUpdates Inserts Updates Deletes OvFlReads OvFlCrtes
TEMP1     0     0   0        ... 0     0           43968   0       0       0         0
```

You can then obtain the information for table space 3 by using the **db2pd -tablespaces** command. Sample output is as follows:

```
Tablespace 3 Configuration:
Type Content PageSz ExtentSz Auto Prefetch BufID FSC NumCntrs MaxStripe LastConsecPg Name
DMS  UsrTmp  4096   32       Yes  32       1     On  1        0         31           TEMPSPACE2

Tablespace 3 Statistics:
TotalPgs UsablePgs UsedPgs PndFreePgs FreePgs HWM  State      MinRecTime NQuiescers
5000     4960      1088    0          3872    1088 0x00000000 0          0

Tablespace 3 Autoresize Statistics:
AS  AR  InitSize    IncSize     IIP MaxSize     LastResize      LRF
No  No  0           0           No  0           None            No

Containers:
ContainNum Type    TotalPgs   UseablePgs StripeSet  Container
0          File    5000       4960       0          /home/db2inst1/tempspace2a
```

The MinRecTime column returns a value that is a UNIX time stamp in a Coordinated Universal Time (UTC) timezone format. To convert the value to a GMT time zone format, you can use the DB2 time stamp function. For example, if MinRecTime returns a value of 1369626329, to convert this value to a GMT format run the following statement:

```
db2 "values timestamp('1970-01-01-00.00.00') + 1369626329 seconds"
```

The query will return a GMT value of 2013-05-27-03.45.29.000000.

The FreePgs column shows that space is filling up. As the free pages value decreases, there is less space available. Notice also that the value for FreePgs plus the value for UsedPgs equals the value of UsablePgs.

Once this is known, you can identify the dynamic SQL statement that is using the table TEMP1 by running the **db2pd -db sample -dyn**:

```
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:13:06

Dynamic Cache:
Current Memory Used           1022197
Total Heap Size               1271398
Cache Overflow Flag           0
Number of References          237
Number of Statement Inserts   32
Number of Statement Deletes   13
Number of Variation Inserts   21
Number of Statements          19

Dynamic SQL Statements:
AnchID StmtUID NumEnv NumVar NumRef NumExe Text
78        1       2     2      3      2      declare global temporary table temp1 ...
253    1       1      1      24     24     insert into session.temp1 values('TEST')
```

Finally, you can map the information from the preceding output to the applications output to identify the application by running **db2pd -db sample -app**.

```
Applications:
AppHandl [nod-index] NumAgents  CoorPid Status      C-AnchID C-StmtUID
501      [000-00501] 1          11246   UOW-Waiting 0        0

L-AnchID L-StmtUID  Appid
253      1          *LOCAL.db2inst1.050202160426
```

You can use the anchor ID (AnchID) value that identified the dynamic SQL statement to identify the associated application. The results show that the last anchor ID (L-AnchID) value is the same as the anchor ID (AnchID) value. You use the results from one run of **db2pd** in the next run of **db2pd**.

The output from **db2pd -agent** shows the number of rows read (in the Rowsread column) and rows written (in the Rowswrtn column) by the application. These values give you an idea of what the application has completed and what the application still has to complete, as shown in the following sample output:

```
AppHandl [nod-index] AgentPid  Priority  Type  DBName
501      [000-00501] 11246     0         Coord SAMPLE

State       ClientPid  Userid   ClientNm Rowsread   Rowswrtn   LkTmOt
Inst-Active 26377      db2inst1 db2bp    22         9588       NotSet
```

You can map the values for AppHandl and AgentPid resulting from running the **db2pd -agent** command to the corresponding values for AppHandl and CoorPiid resulting from running the **db2pd -app** command.

The steps are slightly different if you suspect that an internal temporary table is filling up the table space. You still use **db2pd -tcbstats** to identify tables with large numbers of inserts, however. Following is sample information for an implicit temporary table:

```
TCB Table Information:
TbspaceID TableID PartID MasterTbs MasterTab TableName          SchemaNm ObjClass    DataSize  ...
1         2       n/a    1         2         TEMP (00001,00002) <30>     <JMC Temp   2470      ...
1         3       n/a    1         3         TEMP (00001,00003) <31>     <JMC Temp   2367      ...
1         4       n/a    1         4         TEMP (00001,00004) <30>     <JMC Temp   1872      ...

TCB Table Stats:
TableName          Scans   UDI   PgReorgs   NoChgUpdts Reads   FscrUpdates Inserts ...
TEMP (00001,00002) 0       0     0          0          0       0           43219   ...
TEMP (00001,00003) 0       0     0          0          0       0           42485   ...
TEMP (00001,00004) 0       0     0          0          0       0           0       ...
```

In this example, there are a large number of inserts for tables with the naming convention TEMP (TbspaceID, TableID). These are implicit temporary tables. The values in the SchemaNm column have a naming convention of the value for AppHandl concatenated with the value for SchemaNm, which makes it possible to identify the application doing the work.

You can then map that information to the output from **db2pd -tablespaces** to see the used space for table space 1. Take note of the relationship between the UsedPgs and UsablePgs values in the table space statistics in the following output:

```
Tablespace Configuration:
Id Type Content PageSz ExtentSz Auto Prefetch ... FSC NumCntrs MaxStripe LastConsecPg Name
1  SMS  SysTmp  4096   32       Yes  320      ... On  10       0         31           TEMPSPACE1

Tablespace Statistics:
Id TotalPgs   UsablePgs  UsedPgs    PndFreePgs FreePgs    HWM    State      MinRecTime NQuiescers
1  6516       6516       6516       0          0          0      0x00000000 0          0

Tablespace Autoresize Statistics:
Address            Id    AS  AR  InitSize    IncSize     IIP MaxSize     LastResize  LRF
0x07800000203FB5A0 1     No  No  0           0           No  0           None        No

Containers:
...
```

You can then identify application handles 30 and 31 (because you saw them in the **-tcbstats** output) by using the command **db2pd -app**:

```
Applications:
AppHandl NumAgents CoorPid Status        C-AnchID C-StmtUID L-AnchID L-StmtUID Appid
31       1         4784182 UOW-Waiting   0        0         107      1         ...4142
30       1         8966270 UOW-Executing 107      1         107      1         ...4013
```

Finally, map the information from the preceding output to the Dynamic SQL output obtained by running the **db2pd -dyn** command:

```
Dynamic SQL Statements:
AnchID StmtUID  NumEnv  NumVar  NumRef  NumExe  Text
107    1        1       1       43      43      select c1, c2 from test group by c1,c2
```

Example 8: Monitoring recovery

If you run the command **db2pd -recovery**, the output shows several counters that you can use to verify that recovery is progressing, as shown in the following sample output. The Current Log and Current LSO values provide the log position. The CompletedWork value is the number of bytes completed thus far.

```
Recovery:
Recovery Status     0x00000401
Current Log         S0000005.LOG
Current LSN         0000001F07BC
Current LSO         000002551BEA
Job Type            ROLLFORWARD RECOVERY
Job ID              7
Job Start Time      (1107380474) Wed Feb  2 16:41:14 2005
Job Description     Database Rollforward Recovery
Invoker Type        User
Total Phases        2
Current Phase       1

Progress:
Address            PhaseNum Description StartTime                CompletedWork  TotalWork
0x0000000200667160 1        Forward     Wed Feb  2 16:41:14 2005 2268098 bytes  Unknown
0x0000000200667258 2        Backward    NotStarted               0 bytes        Unknown
```

Example 9: Determining the amount of resources a transaction is using

If you run the command **db2pd -transactions**, the output shows the number of locks, the first log sequence number (LSN), the last LSN, the first LSO, the last

LSO, the log space used, and the space reserved, as shown in the following sample output. This can be useful for understanding the behavior of a transaction.

```
Transactions:
Address            AppHandl [nod-index] TranHdl  Locks  State  Tflag
0x000000022026D980 797      [000-00797] 2        108    WRITE  0x00000000
0x000000022026E600 806      [000-00806] 3        157    WRITE  0x00000000
0x000000022026F280 807      [000-00807] 4        90     WRITE  0x00000000

Tflag2     Firstlsn       Lastlsn        Firstlso        Lastlso
0x00000000 0x0000001A4212 0x0000001C2022 0x000001072262  0x0000010B2C8C
0x00000000 0x000000107320 0x0000001S3462 0x000001057574  0x0000010B3340
0x00000000 0x0000001BC00C 0x0000001X2F03 0x00000107CF0C  0x0000010B2FDE
LogSpace   SpaceReserved  TID            AxRegCnt   GXID
4518       95450          0x000000000451 1          0
6576       139670         0x0000000003E0 1          0
3762       79266          0x000000000472 1          0
```

Example 10: Monitoring log usage

The command **db2pd -logs** is useful for monitoring log usage for a database. By using thePages Written value, as shown in the following sample output, you can determine whether the log usage is increasing:

```
Logs:
Current Log Number            2
Pages Written                 846
Method 1 Archive Status       Success
Method 1 Next Log to Archive  2
Method 1 First Failure        n/a
Method 2 Archive Status       Success
Method 2 Next Log to Archive  2
Method 2 First Failure        n/a

Address             StartLSN         StartLSO       State      Size  Pages  Filename
0x000000023001BF58  0x00000022F032   0x000001B58000 0x00000000 1000  1000   S0000002.LOG
0x000000023001BE98  0x000000000000   0x000001F40000 0x00000000 1000  1000   S0000003.LOG
0x0000000230008F58  0x000000000000   0x000002328000 0x00000000 1000  1000   S0000004.LOG
```

You can identify two types of problems by using this output:

- If the most recent log archive fails, Archive Status is set to a value of Failure. If there is an ongoing archive failure, preventing logs from being archived at all, Archive Status is set to a value of First Failure.
- If log archiving is proceeding very slowly, the Next Log to Archive value is lower than the Current Log Number value. If archiving is very slow, space for active logs might run out, which in turn might prevent any data changes from occurring in the database.

Note: S0000003.LOG and S0000004.LOG do not contain any log records yet and therefore the StartLSN is 0x0

Example 11: Viewing the sysplex list

Without the **db2pd -sysplex** command showing the following sample output, the only other way to report the sysplex list is by using a DB2 trace.

```
Sysplex List:
Alias:         HOST
Location Name: HOST1
Count:         1

IP Address      Port       Priority   Connections Status     PRDID
 1.2.34.56      400        1          0           0
```

Example 12: Generating stack traces

You can use the **db2pd -stack all** command for Windows operating systems or the **-stack** command for UNIX operating systems to produce stack traces for all processes in the current database partition. You might want to use this command iteratively when you suspect that a process or thread is looping or hanging.

You can obtain the current call stack for a particular engine dispatchable unit (EDU) by issuing the command **db2pd -stack** *eduid*, as shown in the following example:

```
Attempting to dump stack trace for eduid 137.
See current DIAGPATH for trapfile.
```

If the call stacks for all of the DB2 processes are desired, use the command **db2pd -stack all**, for example (on Windows operating systems):

```
Attempting to dump all stack traces for instance.
See current DIAGPATH for trapfiles.
```

If you are using a partitioned database environment with multiple physical nodes, you can obtain the information from all of the partitions by using the command **db2_all "; db2pd -stack all"**. If the partitions are all logical partitions on the same machine, however, a faster method is to use **db2pd -alldbp -stacks**.

You can also redirect the output of the **db2pdb -stacks** command for **db2sysc** processes to a specific directory path with the **dumpdir** parameter. The output can be redirected for a specific duration only with the **timeout** parameter. For example, to redirect the output of stack traces for all EDUs in **db2sysc** processes to /home/waleed/mydir for 30 seconds, issue the following command:

```
db2pd -alldbp -stack all dumpdir=/home/waleed/mydir timeout=30
```

Example 13: Viewing memory statistics for a database partition

The **db2pd -dbptnmem** command shows how much memory the DB2 server is currently consuming and, at a high level, which areas of the server are using that memory.

The following example shows the output from running **db2pd -dbptnmem** on an AIX machine:

```
Database Partition Memory Controller Statistics

Controller Automatic: Y
Memory Limit:   122931408 KB
Current usage:     651008 KB
HWM usage:         651008 KB
Cached memory:     231296 KB
```

The descriptions of these data fields and columns are as follows:

Controller Automatic

Indicates the memory controller setting. It shows the value "Y" if the **instance_memory** configuration parameter is set to AUTOMATIC. This means that database manager automatically determines the upper boundary of memory consumption.

Memory Limit

If an instance memory limit is enforced, the value of the **instance_memory** configuration parameter is the upper bound limit of DB2 server memory that can be consumed.

Current usage
The amount of memory the server is currently consuming.

HWM usage
The high water mark (HWM) or peak memory usage that has been consumed since the activation of the database partition (when the **db2start** command was run).

Cached memory
The amount of the current usage that is not currently being used but is cached for performance reasons for future memory requests.

Following is the continuation of the sample output from running **db2pd -dbptnmem** on an AIX operating system:

```
Individual Memory Consumers:
Name          Mem Used (KB)   HWM Used (KB)   Cached (KB)
===========================================================
APPL-DBONE       160000          160000        159616
DBMS-name         38528           38528          3776
FMP_RESOURCES     22528           22528             0
PRIVATE           13120           13120           740
FCM_RESOURCES     10048           10048             0
LCL-p606416         128             128             0
DB-DBONE         406656          406656         67200
```

All registered “consumers” of memory within the DB2 server are listed with the amount of the total memory they are consuming. The column descriptions are as follows:

Name A short, distinguishing name of a consumer of memory, such as the following ones:

APPL-*dbname*
Application memory consumed for database *dbname*

DBMS-*name*
Global database manager memory requirements

FMP_RESOURCES
Memory required to communicate with **db2fmps**

PRIVATE
Miscellaneous private memory requirements

FCM_RESOURCES
Fast Communication Manager resources

LCL-*pid*
The memory segment used to communicate with local applications

DB-*dbname*
Database memory consumed for database *dbname*

Mem Used (KB)
The amount of memory that is currently allotted to the consumer

HWM Used (KB)
The high-water mark (HWM) of the memory, or the peak memory, that the consumer has used

Cached (KB)
Of the Mem Used (KB), the amount of memory that is not currently being used but is immediately available for future memory allocations

Example 14: Monitoring the progress of index reorganization

In DB2 Version 9.8 Fix Pack 3 and later fix packs, the progress report of an index reorganization has the following characteristics:

- The **`db2pd -reorgs index`** command reports index reorg progress for partitioned indexes (Fix Pack 1 introduced support for only non-partitioned indexes).
- The **`db2pd -reorgs index`** command supports the monitoring of index reorg at the partition level (that is, during reorganization of a single partition).
- The reorg progress for non-partitioned and partitioned indexes is reported in separate outputs. One output shows the reorg progress for non-partitioned indexes, and the following outputs show the reorg progress for partitioned indexes on each table partition; the index reorg statistics of only one partition is reported in each output.
- Non-partitioned indexes are processed first, followed by partitioned indexes in serial fashion.
- The **`db2pd -reorgs index`** command displays the following additional information fields in the output for partitioned indexes:
  - `MaxPartition` - Total number of partitions for the table being processed. For partition-level reorg, `MaxPartition` will always have a value of 1 since only a single partition is being reorganized.
  - `PartitionID` - The data partition identifier for the partition being processed.

The following example shows an output obtained using the **`db2pd -reorgs index`** command which reports the index reorg progress for a range-partitioned table with 2 partitions.

Note: The first output reports the Index Reorg Stats of the non-partitioned indexes. The following outputs report the Index Reorg Stats of the partitioned indexes on each partition.

```
Index Reorg Stats:
Retrieval Time: 02/08/2010 23:04:21
TbspaceID: -6       TableID: -32768
Schema: ZORAN    TableName: BIGRPT
Access: Allow none
Status: Completed
Start Time: 02/08/2010 23:03:55   End Time: 02/08/2010 23:04:04
Total Duration: 00:00:08
Prev Index Duration:  -
Cur Index Start: -
Cur Index: 0            Max Index: 2            Index ID: 0
Cur Phase: 0          ( -     )   Max Phase: 0
Cur Count: 0                      Max Count: 0
Total Row Count: 750000

Retrieval Time: 02/08/2010 23:04:21
TbspaceID: 2        TableID: 5
Schema: ZORAN    TableName: BIGRPT
PartitionID: 0      MaxPartition: 2
Access: Allow none
Status: Completed
Start Time: 02/08/2010 23:04:04   End Time: 02/08/2010 23:04:08
Total Duration: 00:00:04
Prev Index Duration:  -
Cur Index Start: -
Cur Index: 0            Max Index: 2            Index ID: 0
Cur Phase: 0          ( -     )   Max Phase: 0
Cur Count: 0                      Max Count: 0
Total Row Count: 375000
```

```
Retrieval Time: 02/08/2010 23:04:21
TbspaceID: 2        TableID: 6
Schema: ZORAN    TableName: BIGRPT
PartitionID: 1      MaxPartition: 2
Access: Allow none
Status: Completed
Start Time: 02/08/2010 23:04:08   End Time: 02/08/2010 23:04:12
Total Duration: 00:00:04
Prev Index Duration:  -
Cur Index Start: -
Cur Index: 0            Max Index: 2            Index ID: 0
Cur Phase: 0          ( -     )   Max Phase: 0
Cur Count: 0                      Max Count: 0
Total Row Count: 375000
```

Example 15: Displaying the top EDUs by processor time consumption and displaying EDU stack information

If you issue the **db2pd** command with the **-edus** parameter option, the output lists all engine dispatchable units (EDUs). Output for EDUs can be returned at the level of granularity you specify, such as at the instance level or at the member. On Linux and UNIX operating systems only, you can also specify the **interval** parameter suboption so that two snapshots of all EDUs are taken, separated by an interval you specify. When the **interval** parameter is specified, two additional columns in the output indicate the delta of processor user time (USR DELTA column) and the delta of processor system time (SYS DELTA column) across the interval.

In the following example, the deltas for processor user time and processor system time are given across a five-second interval:

```
$ db2pd -edus interval=5

Database Partition 0 -- Active -- Up 0 days 00:53:29 -- Date 06/04/2010 03:34:59

List of all EDUs for database partition 0

db2sysc PID: 1249522
db2wdog PID: 2068678

EDU ID TID  Kernel TID EDU Name                         USR        SYS       USR DELTA  SYS DELTA
================================================================================================
6957   6957 13889683   db2agntdp (SAMPLE  ) 0           58.238506  0.820466  1.160726   0.014721
6700   6700 11542589   db2agent (SAMPLE) 0              52.856696  0.754420  1.114821   0.015007
5675   5675 4559055    db2agntdp (SAMPLE  ) 0           60.386779  0.854234  0.609233   0.014304
3088   3088 13951225   db2agntdp (SAMPLE  ) 0           80.073489  2.249843  0.499766   0.006247
3615   3615 2887875    db2loggw (SAMPLE) 0              0.939891   0.410493  0.011694   0.004204
4900   4900 6344925    db2pfchr (SAMPLE) 0              1.748413   0.014378  0.014343   0.000103
7986   7986 13701145   db2agntdp (SAMPLE  ) 0           1.410225   0.025900  0.003636   0.000074
2571   2571 8503329    db2ipccm 0                       0.251349   0.083787  0.002551   0.000857
7729   7729 14168193   db2agntdp (SAMPLE  ) 0           1.717323   0.029477  0.000998   0.000038
7472   7472 11853991   db2agnta (SAMPLE) 0              1.860115   0.032926  0.000860   0.000012
3358   3358 2347127    db2loggr (SAMPLE) 0              0.151042   0.184726  0.000387   0.000458
515    515  13820091   db2aiothr 0                      0.405538   0.312007  0.000189   0.000178
7215   7215 2539753    db2agntdp (SAMPLE  ) 0           1.165350   0.019466  0.000291   0.000008
6185   6185 2322517    db2wlmd (SAMPLE) 0               0.061674   0.034093  0.000169   0.000100
6442   6442 2756793    db2evmli (DB2DETAILDEADLOCK) 0   0.072142   0.052436  0.000092   0.000063
4129   4129 15900799   db2glock (SAMPLE) 0              0.013239   0.000741  0.000064   0.000001
2      2    11739383   db2alarm 0                       0.036904   0.028367  0.000009   0.000009
4386   4386 13361367   db2dlock (SAMPLE) 0              0.015653   0.001281  0.000014   0.000003
1029   1029 15040579   db2fcms 0                        0.041929   0.016598  0.000010   0.000004
5414   5414 14471309   db2pfchr (SAMPLE) 0              0.000093   0.000002  0.000000   0.000000
258    258  13656311   db2sysc 0                        8.369967   0.263539  0.000000   0.000000
5157   5157 7934145    db2pfchr (SAMPLE) 0              0.027598   0.000177  0.000000   0.000000
1543   1543 2670647    db2fcmr 0                        0.004191   0.000079  0.000000   0.000000
1286   1286 8417339    db2extev 0                       0.000312   0.000043  0.000000   0.000000
2314   2314 14360813   db2licc 0                        0.000371   0.000051  0.000000   0.000000
5928   5928 3137537    db2taskd (SAMPLE) 0              0.004903   0.000572  0.000000   0.000000
3872   3872 2310357    db2lfr (SAMPLE) 0                0.000126   0.000007  0.000000   0.000000
4643   4643 11694287   db2pclnr (SAMPLE) 0              0.000094   0.000002  0.000000   0.000000
1800   1800 5800175    db2extev 0                       0.001212   0.002137  0.000000   0.000000
```

```
772    772  7925817    db2thcln 0                       0.000429   0.000072  0.000000   0.000000
2057   2057 6868993    db2pdbc 0                        0.002423   0.001603  0.000000   0.000000
2828   2828 10866809   db2resync 0                      0.016764   0.003098  0.000000   0.000000
```

To provide information only about the EDUs that are the top consumers of processor time and to reduce the amount of output returned, you can further include the **top** parameter option. In the following example, only the top five EDUs are returned, across an interval of 5 seconds. Stack information is also returned, and can be found stored separately in the directory path specified by DUMPDIR, which defaults to **diagpath**.

```
$ db2pd -edus interval=5 top=5 stacks

Database Partition 0 -- Active -- Up 0 days 00:54:00 -- Date 06/04/2010 03:35:30

List of all EDUs for database partition 0

db2sysc PID: 1249522
db2wdog PID: 2068678

EDU ID TID   Kernel TID EDU Name             USR        SYS       USR DELTA  SYS DELTA
=========================================================================================
3358   3358  2347127    db2loggr (SAMPLE) 0  0.154906   0.189223  0.001087   0.001363
3615   3615  2887875    db2loggw (SAMPLE) 0  0.962744   0.419617  0.001779   0.000481
515    515   13820091   db2aiothr 0          0.408039   0.314045  0.000658   0.000543
258    258   13656311   db2sysc 0            8.371388   0.264812  0.000653   0.000474
6700   6700  11542589   db2agent (SAMPLE) 0  54.814420  0.783323  0.000455   0.000310


$ ls -ltr
total 552
drwxrwxr-t    2 vbmithun build         256 05-31 09:59 events/
drwxrwxr-t    2 vbmithun build         256 06-04 03:17 stmmlog/
-rw-r--r--    1 vbmithun build       46413 06-04 03:35 1249522.3358.000.stack.txt
-rw-r--r--    1 vbmithun build       22819 06-04 03:35 1249522.3615.000.stack.txt
-rw-r--r--    1 vbmithun build       20387 06-04 03:35 1249522.515.000.stack.txt
-rw-r--r--    1 vbmithun build       50426 06-04 03:35 1249522.258.000.stack.txt
-rw-r--r--    1 vbmithun build      314596 06-04 03:35 1249522.6700.000.stack.txt
-rw-r--r--    1 vbmithun build       94913 06-04 03:35 1249522.000.processObj.txt
```

Example 16: Displaying agent event metrics

The **db2pd** command supports returning event metrics for agents. If you need to determine whether an agent changed state during a specific period of time, use the event option together with the **-agents** parameter. The AGENT_STATE_LAST_UPDATE_TIME(Tick Value) column that is returned shows the last time that the event being processed by the agent was changed. Together with a previously obtained value for AGENT_STATE_LAST_UPDATE_TIME(Tick Value), you can determine whether an agent has moved on to a new task or whether it continues to process the same task over an extended period of time.

```
db2pd -agents event
Database Partition 0 -- Active -- Up 0 days 03:18:52 -- Date 06/27/2011 11:47:10

Agents:
Current agents:      12
Idle agents:         0
Active coord agents: 10
Active agents total: 10
Pooled coord agents: 2
Pooled agents total: 2

AGENT_STATE_LAST_UPDATE_TIME(Tick Value) EVENT_STATE EVENT_TYPE EVENT_OBJECT EVENT_OBJECT_NAME
2011-06-27-14.44.38.859785(...968075)    IDLE        WAIT       REQUEST      n/a
```

Troubleshooting scripts

You may have internal tools or scripts that are based on the processes running in the database engine. These tools or scripts may no longer work because all agents, prefetchers, and page cleaners are now considered threads in a single, multi-threaded process.

Your internal tools and scripts will have to be modified to account for a threaded process. For example, you may have scripts that start the **ps** command to list the process names; and then perform tasks against certain agent processes. Your scripts must be rewritten.

The problem determination database command **db2pd** will have a new option `-edu` (short for "engine dispatchable unit") to list all agent names along with their thread IDs. The **db2pd -stack** command continues to work with the threaded engine to dump individual EDU stacks or to dump all EDU stacks for the current node.

db2dart command

The **db2dart** command can be used to verify the architectural correctness of databases and the objects within them. It can also be used to display the contents of database control files in order to extract data from tables that might otherwise be inaccessible.

To display all of the possible options, issue the **db2dart** command without any parameters. Some options that require parameters, such as the table space ID, are prompted for if they are not explicitly specified on the command line.

By default, the **db2dart** utility will create a report file with the name `databaseName.RPT`. For single-partition database partition environments, the file is created in the current directory. For multiple-partition database partition environments, the file is created under a subdirectory in the diagnostic directory. The subdirectory is called `DART####`, where `####` is the database partition number.

In a DB2 pureScale environment, some metadata files (such as bufferpool configuration files) exist for each member and are validated or updated on a per-member basis (rather than per-database).

The **db2dart** utility accesses the data and metadata in a database by reading them directly from disk. Because of that, you should never run the tool against a database that still has active connections. If there are connections, the tool will not know about pages in the buffer pool or control structures in memory, for example, and might report false errors as a result. Similarly, if you run **db2dart** against a database that requires crash recovery or that has not completed rollforward recovery, similar inconsistencies might result due to the inconsistent nature of the data on disk.

Comparison of INSPECT and db2dart

The **INSPECT** command inspects a database for architectural integrity, checking the pages of the database for page consistency. The **INSPECT** command checks that the structures of table objects and structures of table spaces are valid. Cross object validation conducts an online index to data consistency check. The **db2dart** command examines databases for architectural correctness and reports any encountered errors.

The **INSPECT** command is similar to the **db2dart** command in that it allows you to check databases, table spaces, and tables. A significant difference between the two commands is that the database needs to be deactivated before you run **db2dart**, whereas INSPECT requires a database connection and can be run while there are simultaneous active connections to the database.

If you do not deactivate the database, **db2dart** will yield unreliable results.

The following tables list the differences between the tests that are performed by the **db2dart** and **INSPECT** commands.

Table 92. Feature comparison of db2dart and INSPECT for table spaces

Tests performed	db2dart	INSPECT
SMS table spaces		
Check table space files	YES	NO
Validate contents of internal page header fields	YES	YES
DMS table spaces		
Check for extent maps pointed at by more than one object	YES	NO
Check every extent map page for consistency bit errors	NO	YES
Check every space map page for consistency bit errors	NO	YES
Validate contents of internal page header fields	YES	YES
Verify that extent maps agree with table space maps	YES	NO

Table 93. Feature comparison of db2dart and INSPECT for data objects

Tests performed	db2dart	INSPECT
Check data objects for consistency bit errors	YES	YES
Check the contents of special control rows	YES	NO
Check the length and position of variable length columns	YES	NO
Check the LONG VARCHAR, LONG VARGRAPHIC, and large object (LOB) descriptors in table rows	YES	NO
Check the summary total pages, used pages and free space percentage	NO	YES
Validate contents of internal page header fields	YES	YES

Table 93. Feature comparison of db2dart and INSPECT for data objects (continued)

Tests performed	db2dart	INSPECT
Verify each row record type and its length	YES	YES
Verify that rows are not overlapping	YES	YES

Table 94. Feature comparison of db2dart and INSPECT for index objects

Tests performed	db2dart	INSPECT
Check for consistency bit errors	YES	YES
Check the location and length of the index key and whether there is overlapping	YES	YES
Check the ordering of keys in the index	YES	NO
Determine the summary total pages and used pages	NO	YES
Validate contents of internal page header fields	YES	YES
Verify the uniqueness of unique keys	YES	NO
Check for the existence of the data row for a given index entry	NO	YES
Verify each key to a data value	NO	YES

Table 95. Feature comparison of db2dart and INSPECT for block map objects

Tests performed	db2dart	INSPECT
Check for consistency bit errors	YES	YES
Determine the summary total pages and used pages	NO	YES
Validate contents of internal page header fields	YES	YES

Table 96. Feature comparison of db2dart and INSPECT for long field and LOB objects

Tests performed	db2dart	INSPECT
Check the allocation structures	YES	YES
Determine the summary total pages and used pages (for LOB objects only)	NO	YES

In addition, the following actions can be performed using the **db2dart** command:

- Format and dump data pages
- Format and dump index pages

- Format data rows to delimited ASCII
- Mark an index invalid

The **INSPECT** command cannot be used to perform those actions.

db2val command

The db2val command verifies the basic functions of a DB2 copy by checking the state of installation files, instance setup, and local database connections.

Validating your DB2 copy

Use the db2val command to determine whether your DB2 copy is functioning properly.

About this task

The **db2val** tool verifies the core function of a DB2 copy by validating installation files, instances, database creation, connections to that database, and the state of partitioned database environments. This validation can be helpful if you have manually deployed a DB2 copy on Linux and UNIX operating systems using `tar.gz` files. The **db2val** command can quickly ensure that all the configuration has been correctly done and ensure that the DB2 copy is what you expect it to be. You can specify instances and databases or you can run **db2val** against all of the instances. The **db2val** command can be found in the *DB2-install-path*`\bin` and `sqllib/bin` directories.

Example

For example, to validate all the instances for the DB2 copy, run the following command:

```
db2val -a
```

For complete **db2val** command details and further example, refer to the "db2val - DB2 copy validation tool command" topic.

Part 5. DB2 commands for database administration

Version 10.1 introduces changes to DB2 CLP commands, DB2 system commands, and SQL statements to support new capabilities. These changes can affect your existing database applications or database administration scripts.

Chapter 32. Data movement options

There are various data movement options available in DB2 for Linux, UNIX, and Windows. This topic provides an overview of the data movement tools, utilities, stored procedures, and commands available to you.

Use these tables as a guide to help you determine which data movement options might best suit your needs.

Table 97. Load utility

Method	Load utility
Purpose	To efficiently move large quantities of data into newly created tables, or into tables that already contain data.
Cross platform compatible	Yes
Best practice usage	This utility is best suited to situations where performance is your primary concern. This utility can be used as an alternative to the import utility. It is faster then the import utility because it writes formatted pages directly into the database rather than using SQL INSERTS. In addition, the load utility allows you the option to not log the data or use the `COPY` option to save a copy of the loaded data. Load operations can fully exploit resources, such as CPUs and memory on SMP and MPP environments.
References	Loading data

Table 98. Ingest utility

Method	Ingest utility
Purpose	Streams data from files and pipes into DB2 target tables, while still keeping those tables available.
Cross platform compatible	Yes
Best practice usage	This utility strikes a good balance between performance and availability, but if the latter is more important to you, then you should choose the ingest utility instead of the load utility. Similar to the import utility, ingest is suitable if the target tables are updatable views, range-clustered tables, or nicknames; however, the ingest utility has superior performance.
References	Ingesting data

Table 99. Import utility

Method	Import utility
Purpose	To insert data from an external file into a table, hierarchy, view, or nickname
Cross platform compatible	Yes

Table 99. Import utility (continued)

Best practice usage	The import utility can be a good alternative to the load utility in the following situations: • where the target table is a view • the target table has constraints and you don't want the target table to be put in the Set Integrity Pending state • the target table has triggers and you want them fired
References	Importing data

Table 100. Export utility

Method	Export utility
Purpose	To export data from a database to one of several external file formats. The data can then be imported or loaded at a later time.
Cross platform compatible	Yes
Best practice usage	This utility is best suited in situations where you want to store data in an external file, to either process it further or move data to another table. High Performance Unload (HPU) is an alternative, however, it must be purchased separately. Export supports XML columns.
References	Exporting data

Table 101. db2move command

Method	**db2move** command
Purpose	Using the **db2move** utility with the COPY option, allows you to copy schema templates (with or without data) from a source database to a target database or move an entire schema from a source database to a target database. Using the **db2move** utility with the IMPORT or EXPORT option facilitates the movement of a large numbers of tables between DB2 databases.
Cross platform compatible	Yes
Best practice usage	When used with the COPY option, the source and the target database must be different. The COPY option is useful in making schema templates. Use the IMPORT or EXPORT option for cloning databases when there is no support for cross-platform backup and restore operations. The IMPORT and EXPORT options are used in conjunction with the **db2look** command.
References	• “Copying a schema” in *Database Administration Concepts and Configuration Reference*

Table 102. RESTORE command

Method	**RESTORE** command with the REDIRECT option and the GENERATE SCRIPT option
Purpose	To copy an entire database from one system to another using a script from an existing backup image.
Cross platform compatible	Limited. See References
Best practice usage	This utility is best suited in situations where a backup image exists.

Table 102. RESTORE command (continued)

References	• “Performing a redirected restore using an automatically generated script” in *Data Recovery and High Availability Guide and Reference* • “Backup and restore operations between different operating systems and hardware platforms” in *Data Recovery and High Availability Guide and Reference*

Table 103. db2relocatedb command

Method	**db2relocatedb** command
Purpose	To rename a database, or relocate a database or part of a database to the same system or a different system.
Cross platform compatible	No
Best practice usage	• This utility can be used for situations where a backup and restore could be time consuming. • This utility is an alternative to using backup and restore to move or create copies of databases. • It also provides a quick method of cloning a database for alternative environments such as testing. • It can be used to move table space containers to a new set of storage devices
References	“db2relocatedb - Relocate database command” in *Command Reference*

Table 104. ADMIN_COPY_SCHEMA procedure

Method	ADMIN_COPY_SCHEMA procedure
Purpose	Allows you to make a copy of all the objects in a single schema and re-create those objects in a new schema. This copy operation can be performed with or without data, within a database.
Cross platform compatible	Yes
Best practice usage	This utility is useful for making schema templates. It is also useful if you want to experiment with a schema (for example, try out new indexes) without impacting the source schema's behavior. The key differences between the ADMIN_COPY_SCHEMA procedure and the **db2move** utility are: • The ADMIN_COPY_SCHEMA procedure is used on a single database while the **db2move** utility is used across databases • The **db2move** utility fails when invoked if it cannot create a physical object such as a table or index. The ADMIN_COPY_SCHEMA procedure logs errors and continues. • The ADMIN_COPY_SCHEMA procedure uses load from cursor to move data from one schema to the other. The **db2move** utility uses a remote load, similar to a load from cursor, which pulls in the data from the source database.
References	“Copying a schema” in *Database Administration Concepts and Configuration Reference*

Table 105. ADMIN_MOVE_TABLE procedure

Method	ADMIN_MOVE_TABLE procedure
Purpose	Allows you to move the data in a table to a new table object of the same name (but with possibly different storage characteristics) while the data remains online and available for access.
Cross platform compatible	Yes
Best practice usage	This utility automates the process of moving table data to a new table object while allowing the data to remain online for select, insert, update, and delete access. You can also generate a compression dictionary when a table is moved. • Avoid making multiple moves into same table space at the same time. • Run this procedure when activity on the table is low. • Use a multi-step move operation. The INIT and COPY phases can be called at any time. Execute the REPLAY phase multiple times in order to keep the staging table size small, and then issue the SWAP during a time of low activity on the table. • Consider using an offline table move if you are working with tables without unique indexes or tables with no index.
References	• "ADMIN_MOVE_TABLE procedure - Move an online table" in *Command Reference* • Moving tables online by using the ADMIN_MOVE_TABLE procedure

Table 106. Split mirror

Method	Split mirror
Purpose	To create a clone, standby, or backup database
Cross platform compatible	No
Best practice usage	• create a standby system in case of a primary failure to reduce down time • move backup operations away from a live production machine onto a split database • provides a quick method of cloning a database for alternate environments, such as testing
Considerations	• only DMS table spaces can be backed up on the split version of the database • usually used in conjunction with some flashcopy technology provided with storage systems • an alternative is to issue a file copy once the database is suspended, however this duplicates the amount of storage for the database
References	"High availability through online split mirror and suspended I/O support" in *Data Recovery and High Availability Guide and Reference*

Chapter 33. Load utility

The load utility is capable of efficiently moving large quantities of data into newly created tables, or into tables that already contain data.

The utility can handle most data types, including XML, large objects (LOBs), and user-defined types (UDTs).

The load utility is faster than the import utility, because it writes formatted pages directly into the database, while the import utility performs SQL INSERTs.

The load utility does not fire triggers, and does not perform referential or table constraints checking (other than validating the uniqueness of the indexes).

The load process consists of four distinct phases (see Figure 44):

1. **Load**
   During the load phase, data is loaded into the table, and index keys and table statistics are collected, if necessary. *Save points*, or points of consistency, are established at intervals specified through the **SAVECOUNT** parameter in the **LOAD** command. Messages are generated, indicating how many input rows were successfully loaded at the time of the save point.
2. **Build**
   During the build phase, indexes are produced based on the index keys collected during the load phase. The index keys are sorted during the load phase, and index statistics are collected (if the **STATISTICS USE PROFILE** option was specified, and profile indicates collecting index statistics). The statistics are similar to those collected through the **RUNSTATS** command.
3. **Delete**
   During the delete phase, the rows that caused a unique or primary key violation are removed from the table. These deleted rows are stored in the load exception table, if one was specified.
4. **Index copy**
   During the index copy phase, the index data is copied from a system temporary table space to the original table space. This will only occur if a system temporary table space was specified for index creation during a load operation with the **READ ACCESS** option specified.

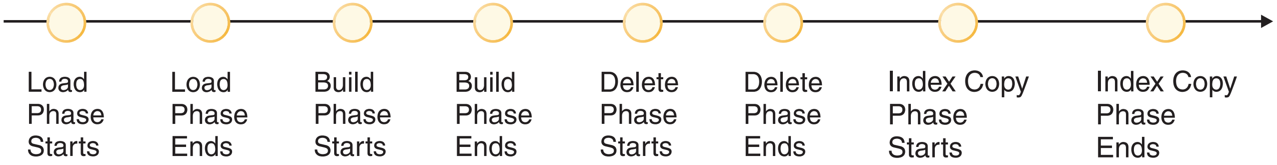


Figure 44. The Four Phases of the Load Process: Load, Build, Delete, and Index Copy

Note: After you invoke the load utility, you can use the **LIST UTILITIES** command to monitor the progress of the load operation.

The following information is required when loading data:

- The path and the name of the input file, named pipe, or device.
- The name or alias of the target table.

- The format of the input source. This format can be DEL, ASC, PC/IXF, or CURSOR.
- Whether the input data is to be appended to the table, or is to replace the existing data in the table.
- A message file name, if the utility is invoked through the application programming interface (API), db2Load.

Load modes

- **INSERT**
  In this mode, load appends input data to the table without making any changes to the existing data.
- **REPLACE**
  In this mode, load deletes existing data from the table and populates it with the input data.
- **RESTART**
  In this mode, an interrupted load is resumed. In most cases, the load is resumed from the phase it failed in. If that phase was the load phase, the load is resumed from the last successful consistency point.
- **TERMINATE**
  In this mode, a failed load operation is rolled back.

The options you can specify include:

- That the data to be loaded resides on the client, if the load utility is invoked from a remotely connected client. Note that XML and LOB data are always read from the server, even you specify the **CLIENT** option.
- The method to use for loading the data: column location, column name, or relative column position.
- How often the utility is to establish consistency points.
- The names of the table columns into which the data is to be inserted.
- Whether or not preexisting data in the table can be queried while the load operation is in progress.
- Whether the load operation should wait for other utilities or applications to finish using the table or force the other applications off before proceeding.
- An alternate system temporary table space in which to build the index.
- The paths and the names of the input files in which LOBs are stored.

  Note: The load utility does not honor the **COMPACT** lob option.
- A message file name. During load operations, you can specify that message files be created to contain the error, warning, and informational messages associated with those operations. Specify the name of these files with the MESSAGES parameter.

  Note:

  1. You can only view the contents of a message file after the operation is finished. If you want to view load messages while a load operation is running, you can use the **LOAD QUERY** command.
  2. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility.
- Whether column values being loaded have implied decimal points.
- Whether the utility should modify the amount of free space available after a table is loaded.

- Whether statistics are to be gathered during the load process. This option is only supported if the load operation is running in REPLACE mode. Statistics are collected according to the profile defined for the table. The profile must be created by the **RUNSTATS** command before the **LOAD** command is executed. If the profile does not exist and the load operation is instructed to collect statistics according to the profile, the load will fail, and an error message will be returned.

  If data is appended to a table, statistics are not collected. To collect current statistics on an appended table, invoke the **RUNSTATS** utility following completion of the load process. If gathering statistics on a table with a unique index, and duplicate keys are deleted during the delete phase, statistics are not updated to account for the deleted records. If you expect to have a significant number of duplicate records, do not collect statistics during the load operation. Instead, invoke the **RUNSTATS** utility following completion of the load process.
- Whether to keep a copy of the changes made. This is done to enable rollforward recovery of the database. This option is not supported if rollforward recovery is disabled for the database; that is, if the database configuration parameters **logarchmeth1** and **logarchmeth2** are set to OFF. If no copy is made, and rollforward recovery is enabled, the table space is left in Backup Pending state at the completion of the load operation.

  Logging is required for fully recoverable databases. The load utility almost completely eliminates the logging associated with the loading of data. In place of logging, you have the option of making a copy of the loaded portion of the table. If you have a database environment that allows for database recovery following a failure, you can do one of the following:
  - Explicitly request that a copy of the loaded portion of the table be made.
  - Take a backup of the table spaces in which the table resides immediately after the completion of the load operation.

  If the database configuration parameter **logindexbuild** is set, and if the load operation is invoked with the **COPY YES** recoverability option and the INCREMENTAL indexing option, the load logs all index modifications. The benefit of using these options is that when you roll forward through the log records for this load, you also recover the indexes (whereas normally the indexes are not recovered unless the load uses the REBUILD indexing mode).

  If you are loading a table that already contains data, and the database is non-recoverable, ensure that you have a backed-up copy of the database, or the table spaces for the table being loaded, before invoking the load utility, so that you can recover from errors.

  If you want to perform a sequence of multiple load operations on a recoverable database, the sequence of operations will be faster if you specify that each load operation is non-recoverable, and take a backup at the end of the load sequence, than if you invoke each of the load operations with the **COPY YES** option. You can use the NONRECOVERABLE option to specify that a load transaction is to be marked as non-recoverable, and that it will not be possible to recover it by a subsequent rollforward operation. The rollforward utility will skip the transaction, and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the rollforward operation is completed, such a table can only be dropped (see Figure 45 on page 552). With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation.

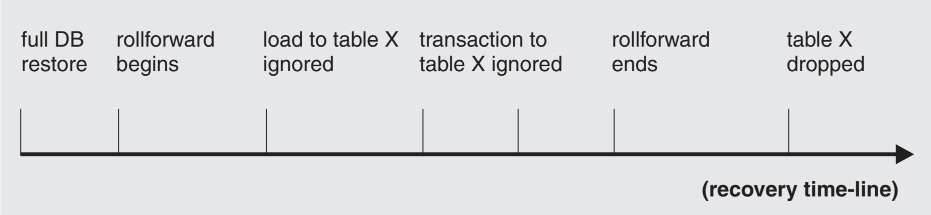


Figure 45. Non-recoverable Processing During a Roll Forward Operation

- The fully qualified path to be used when creating temporary files during a load operation. The name is specified by the **TEMPFILES PATH** parameter of the **LOAD** command. The default value is the database path. The path resides on the server machine, and is accessed by the DB2 instance exclusively. Therefore, any path name qualification given to this parameter must reflect the directory structure of the server, not the client, and the DB2 instance owner must have read and write permission on the path.

Privileges and authorities required to use load

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

To load data into a table, you must have one of the following:

- DATAACCESS authority
- LOAD or DBADM authority on the database and
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.
  - SELECT privilege on SYSCAT.TABLES is required in some cases where LOAD queries the catalog tables.

Since all load processes (and all DB2 server processes, in general), are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

If the REPLACE option is specified, the session authorization ID must have the authority to drop the table.

On Windows, and Windows.NET operating systems where DB2 is running as a Windows service, if you are loading data from files that reside on a network drive, you must configure the DB2 service to run under a user account that has read access to these files.

Note:

- To load data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table.
- To load data into a table that has protected rows, the session authorization ID must have been granted a security label for write access that is part of the security policy protecting the table.

LOAD authority

Users having LOAD authority at the database level, as well as INSERT privilege on a table, can use the **LOAD** command to load data into a table.

Note: Having DATAACCESS authority gives a user full access to the LOAD command.

Users having LOAD authority at the database level, as well as INSERT privilege on a table, can **LOAD RESTART** or **LOAD TERMINATE** if the previous load operation is a load to insert data.

Users having LOAD authority at the database level, as well as the INSERT and DELETE privileges on a table, can use the **LOAD REPLACE** command.

If the previous load operation was a load replace, the DELETE privilege must also have been granted to that user before the user can **LOAD RESTART** or **LOAD TERMINATE**.

If the exception tables are used as part of a load operation, the user must have INSERT privilege on the exception tables.

The user with this authority can perform **QUIESCE TABLESPACES FOR TABLE**, **RUNSTATS**, and **LIST TABLESPACES** commands.

Loading data

The load utility efficiently moves large quantities of data into newly created tables, or into tables that already contain data.

Before you begin

Before invoking the load utility, you must be connected to (or be able to implicitly connect to) the database into which you want to load the data.

Since the utility issues a COMMIT statement, complete all transactions and release all locks by issuing either a COMMIT or a ROLLBACK statement before invoking the load utility.

Data is loaded in the sequence that appears in the input file, except when using multidimensional clustering (MDC) tables, partitioned tables, or the **anyorder** file type modifier. If you want a particular sequence, sort the data before attempting a load operation. If clustering is required, the data should be sorted on the clustering index before loading. When loading data into multidimensional clustered tables (MDC), sorting is not required before the load operation, and data is clustered according to the MDC table definition. When loading data into partitioned tables, sorting is not required before the load operation, and data is partitioned according to the table definition.

Restrictions

These are some of the restrictions that apply to the load utility:

- Loading data into nicknames is not supported.
- Loading data into typed tables, or tables with structured type columns, is not supported.
- Loading data into declared temporary tables and created temporary tables is not supported.
- XML data can only be read from the server side; if you want to have the XML files read from the client, use the import utility.
- You cannot create or drop tables in a table space that is in Backup Pending state.
- You cannot load data into a database accessed through DB2 Connect or a server level before DB2 Version 2. Options that are only available with the current cannot be used with a server from the previous release.
- If an error occurs during a **LOAD REPLACE** operation, the original data in the table is lost. Retain a copy of the input data to allow the load operation to be restarted.
- Triggers are not activated on newly loaded rows. Business rules associated with triggers are not enforced by the load utility.
- Loading encrypted data is not supported.

These are some of the restrictions that apply to the load utility when loading into a partitioned table:

- Consistency points are not supported when the number of partitioning agents is greater than one.
- Loading data into a subset of data partitions while keeping the remaining data partitions fully online is not supported.
- The exception table used by a load operation or a set integrity pending operation cannot be partitioned.
- A unique index cannot be rebuilt when the load utility is running in insert mode or restart mode, and the load target table has any detached dependents.

Procedure

To invoke the load utility:

- Issue a **LOAD** command in the command line processor (CLP).
- Call the db2Load application programming interface (API) from a client application.
- Open the task assistant in IBM Data Studio for the **LOAD** command.

Example

The following is an example of a **LOAD** command issued through the CLP:

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   insert into userid.staff copy yes use tsm data buffer 4000
```

In this example:

- Any warning or error messages are placed in the `staff.msgs` file.
- A copy of the changes made is stored in Tivoli Storage Manager (TSM).
- 4000 pages of buffer space are to be used during the load operation.

The following is another example of a **LOAD** command issued through the CLP:

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   tempfiles path /u/myuser replace into staff
```

In this example:

- The table data is being replaced.
- The **TEMPFILES PATH** parameter is used to specify /u/myuser as the server path into which temporary files are written.

Note: These examples use relative path names for the load input file. Relative path names are only allowed on calls from a client on the same database partition as the database. The use of fully qualified path names is recommended.

What to do next

After you invoke the load utility, you can use the **LIST UTILITIES** command to monitor the progress of the load operation. If a load operation is performed in either **INSERT** mode, **REPLACE** mode, or **RESTART** mode, detailed progress monitoring support is available. Issue the **LIST UTILITIES** command with the **SHOW DETAILS** parameter to view detailed information about the current load phase. Details are not available for a load operation performed in **TERMINATE** mode. The **LIST UTILITIES** command simply shows that a load terminate utility is currently running.

A load operation maintains unique constraints, range constraints for partitioned tables, generated columns, and LBAC security rules. For all other constraints, the table is placed in the Set Integrity Pending state at the beginning of a load operation. After the load operation is complete, the SET INTEGRITY statement must be used to take the table out of Set Integrity Pending state.

Load sessions - CLP examples

Example 1

TABLE1 has 5 columns:

- COL1 VARCHAR 20 NOT NULL WITH DEFAULT
- COL2 SMALLINT
- COL3 CHAR 4
- COL4 CHAR 2 NOT NULL WITH DEFAULT
- COL5 CHAR 2 NOT NULL

ASCFILE1 has 6 elements:

- ELE1 positions 01 to 20
- ELE2 positions 21 to 22
- ELE3 positions 23 to 23
- ELE4 positions 24 to 27
- ELE5 positions 28 to 31
- ELE6 positions 32 to 32
- ELE7 positions 33 to 40

Data Records:

```
1...5...10...15...20...25...30...35...40
Test data 1         XXN 123abcdN
Test data 2 and 3   QQY    XXN
Test data 4,5 and 6 WWN6789    Y
```

The following command loads the table from the file:

```
db2 load from ascfile1 of asc modified by striptblanks reclen=40
   method L (1 20, 21 22, 24 27, 28 31)
   null indicators (0,0,23,32)
   insert into table1 (col1, col5, col2, col3)
```

Note:

1. The specification of `striptblanks` in the MODIFIED BY parameter forces the truncation of blanks in VARCHAR columns (COL1, for example, which is 11, 17 and 19 bytes long, in rows 1, 2 and 3, respectively).
2. The specification of `reclen=40` in the MODIFIED BY parameter indicates that there is no newline character at the end of each input record, and that each record is 40 bytes long. The last 8 bytes are not use to load the table.
3. Since COL4 is not provided in the input file, it will be inserted into TABLE1 with its default value (it is defined NOT NULL WITH DEFAULT).
4. Positions 23 and 32 are used to indicate whether COL2 and COL3 of TABLE1 will be loaded NULL for a given row. If there is a `Y` in the column's null indicator position for a given record, the column will be NULL. If there is an `N`, the data values in the column's data positions of the input record (as defined in L(........)) are used as the source of column data for the row. In this example, neither column in row 1 is NULL; COL2 in row 2 is NULL; and COL3 in row 3 is NULL.
5. In this example, the NULL INDICATORS for COL1 and COL5 are specified as 0 (zero), indicating that the data is not nullable.
6. The NULL INDICATOR for a given column can be anywhere in the input record, but the position must be specified, and the `Y` or `N` values must be supplied.

Example 2 (using dump files)

Table FRIENDS is defined as:

```
table friends "( c1 INT NOT NULL, c2 INT, c3 CHAR(8) )"
```

If an attempt is made to load the following data records into this table,

```
23, 24, bobby
, 45, john
4,, mary
```

the second row is rejected because the first INT is NULL, and the column definition specifies NOT NULL. Columns which contain initial characters that are not consistent with the DEL format will generate an error, and the record will be rejected. Such records can be written to a dump file.

DEL data appearing in a column outside of character delimiters is ignored, but does generate a warning. For example:

```
22,34,"bob"
24,55,"sam" sdf
```

The utility will load "sam" in the third column of the table, and the characters "sdf" will be flagged in a warning. The record is not rejected. Another example:

```
22 3, 34,"bob"
```

The utility will load `22,34,"bob"`, and generate a warning that some data in column one following the 22 was ignored. The record is not rejected.

Example 3 (Loading a table with an identity column)

TABLE1 has 4 columns:

- C1 VARCHAR(30)
- C2 INT GENERATED BY DEFAULT AS IDENTITY
- C3 DECIMAL(7,2)
- C4 CHAR(1)

TABLE2 is the same as TABLE1, except that C2 is a GENERATED ALWAYS identity column.

Data records in DATAFILE1 (DEL format):

```
"Liszt"
"Hummel",,187.43, H
"Grieg",100, 66.34, G
"Satie",101, 818.23, I
```

Data records in DATAFILE2 (DEL format):

```
"Liszt", 74.49, A
"Hummel", 0.01, H
"Grieg", 66.34, G
"Satie", 818.23, I
```

Note:

1. The following command generates identity values for rows 1 and 2, since no identity values are supplied in DATAFILE1 for those rows. Rows 3 and 4, however, are assigned the user-supplied identity values of 100 and 101, respectively.

   ```
   db2 load from datafile1.del of del replace into table1
   ```

2. To load DATAFILE1 into TABLE1 so that identity values are generated for all rows, issue one of the following commands:

   ```
   db2 load from datafile1.del of del method P(1, 3, 4)
      replace into table1 (c1, c3, c4)
   db2load from datafile1.del of del modified by identityignore
      replace into table1
   ```

3. To load DATAFILE2 into TABLE1 so that identity values are generated for each row, issue one of the following commands:

   ```
   db2 load from datafile2.del of del replace into table1 (c1, c3, c4)
   db2 load from datafile2.del of del modified by identitymissing
      replace into table1
   ```

4. To load DATAFILE1 into TABLE2 so that the identity values of 100 and 101 are assigned to rows 3 and 4, issue the following command:

   ```
   db2 load from datafile1.del of del modified by identityoverride
      replace into table2
   ```

   In this case, rows 1 and 2 will be rejected, because the utility has been instructed to override system-generated identity values in favor of user-supplied values. If user-supplied values are not present, however, the row must be rejected, because identity columns are implicitly not NULL.

5. If DATAFILE1 is loaded into TABLE2 without using any of the identity-related file type modifiers, rows 1 and 2 will be loaded, but rows 3 and 4 will be rejected, because they supply their own non-NULL values, and the identity column is GENERATED ALWAYS.

Example 3 (loading from CURSOR)

MY.TABLE1 has 3 columns:

- ONE INT
- TWO CHAR(10)
- THREE DATE

MY.TABLE2 has 3 columns:

- ONE INT
- TWO CHAR(10)
- THREE DATE

Cursor MYCURSOR is defined as follows:

```
declare mycursor cursor for select * from my.table1
```

The following command loads all the data from MY.TABLE1 into MY.TABLE2:

```
load from mycursor of cursor method P(1,2,3) insert into
  my.table2(one,two,three)
```

Note:

1. Only one cursor name can be specified in a single LOAD command. That is, `load from mycurs1, mycurs2 of cursor...` is not allowed.
2. `P` and `N` are the only valid METHOD values for loading from a cursor.
3. In this example, METHOD P and the insert column list `(one,two,three)` could have been omitted since they represent default values.
4. MY.TABLE1 can be a table, view, alias, or nickname.

LBAC-protected data load considerations

For a successful load operation into a table with protected rows, you must have LBAC (label-based access control) credentials. You must also provide a valid security label, or a security label that can be converted to a valid label, for the security policy currently associated with the target table.

If you do not have valid LBAC credentials, the load fails and an error (SQLSTATE 42512) is returned. In cases where the input data does not contain a security label or that security label is not in its internal binary format, you can use several file type modifiers to allow your load to proceed.

When you load data into a table with protected rows, the target table has one column with a data type of DB2SECURITYLABEL. If the input row of data does not contain a value for that column, that row is rejected unless the `usedefaults` file type modifier is specified in the load command, in which case the security label you hold for write access from the security policy protecting the table is used. If you do not hold a security label for write access, the row is rejected and processing continues on to the next row.

When you load data into a table that has protected rows and the input data does include a value for the column with a data type of DB2SECURITYLABEL, the same rules are followed as when you insert data into that table. If the security label protecting the row being loaded (the one in that row of the data file) is one that you are able to write to, then that security label is used to protect the row. (In other words, it is written to the column that has a data type of DB2SECURITYLABEL.) If you are not able to write to a row protected by that security label, what happens depends on how the security policy protecting the source table was created:

- If the CREATE SECURITY POLICY statement that created the policy included the option `RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL`, the row is rejected.
- If the CREATE SECURITY POLICY statement did not include the option or if it instead included the `OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL` option, the security label in the data file for that row is ignored and the security label you hold for write access is used to protect that row. No error or warning is issued in this case. If you do not hold a security label for write access, the row is rejected and processing continues on to the next row.

Delimiter considerations

When loading data into a column with a data type of DB2SECURITYLABEL, the value in the data file is assumed by default to be the actual bytes that make up the internal representation of that security label. However, some raw data might contain newline characters which could be misinterpreted by the **LOAD** command as delimiting the row. If you have this problem, use the `delprioritychar` file type modifier to ensure that the character delimiter takes precedence over the row delimiter. When you use `delprioritychar`, any record or column delimiters that are contained within character delimiters are not recognized as being delimiters. Using the `delprioritychar` file type modifier is safe to do even if none of the values contain a newline character, but it does slow the load down slightly.

If the data being loaded is in ASC format, you might have to take an extra step in order to prevent any trailing white space from being included in the loaded security labels and security label names. ASCII format uses column positions as delimiters, so this might occur when loading into variable-length fields. Use the `striptblanks` file type modifier to truncate any trailing blank spaces.

Nonstandard security label values

You can also load data files in which the values for the security labels are strings containing the values of the components in the security label, for example, *S:(ALPHA,BETA)*. To do so you must use the file type modifier `seclabelchar`. When you use `seclabelchar`, a value for a column with a data type of DB2SECURITYLABEL is assumed to be a string constant containing the security label in the string format for security labels. If a string is not in the proper format, the row is not inserted and a warning (SQLSTATE 01H53) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table, the row is not inserted and a warning (SQLSTATE 01H53) is returned.

You can also load a data file in which the values of the security label column are security label names. To load this sort of file you must use the file type modifier `seclabelname`. When you use `seclabelname`, all values for columns with a data type of DB2SECURITYLABEL are assumed to be string constants containing the names of existing security labels. If no security label exists with the indicated name for the security policy protecting the table, the row is not loaded and a warning (SQLSTATE 01H53) is returned.

Rejected rows

Rows that are rejected during the load are sent to either a dumpfile or an exception table (if they are specified in the **LOAD** command), depending on the reason why the rows were rejected. Rows that are rejected due to parsing errors are sent to the dumpfile. Rows that violate security policies are sent to the exception table.

Note: You cannot specify an exception table if the target table contains an XML column.

Examples

For all examples, the input data file `myfile.del` is in DEL format. All are loading data into a table named REPS, which was created with this statement:

```
create table reps (row_label db2securitylabel,
id integer,
name char(30))
security policy data_access_policy
```

For this example, the input file is assumed to contain security labels in the default format:

```
db2 load from myfile.del of del modified by delprioritychar insert into reps
```

For this example, the input file is assumed to contain security labels in the security label string format:

```
db2 load from myfile.del of del modified by seclabelchar insert into reps
```

For this example, the input file is assumed to contain security labels names for the security label column:

```
db2 load from myfile.del of del modified by seclabelname insert into reps
```

Identity column load considerations

The load utility can be used to load data into a table containing an identity column whether or not the input data has identity column values.

If no identity-related file type modifiers are used, the utility works according to the following rules:

- If the identity column is GENERATED ALWAYS, an identity value is generated for a table row whenever the corresponding row in the input file is missing a value for the identity column, or a NULL value is explicitly given. If a non-NULL value is specified for the identity column, the row is rejected (SQL3550W).
- If the identity column is GENERATED BY DEFAULT, the load utility makes use of user-supplied values, if they are provided; if the data is missing or explicitly NULL, a value is generated.

The load utility does not perform any extra validation of user-supplied identity values beyond what is normally done for values of the identity column's data type (that is, SMALLINT, INT, BIGINT, or DECIMAL). Duplicate values are not reported.

In most cases the load utility cannot guarantee that identity column values are assigned to rows in the same order that these rows appear in the data file. Because the assignment of identity column values is managed in parallel by the load utility, those values are assigned in arbitrary order. The exceptions to this are as follows:

- In single-partition databases, rows are not processed in parallel when CPU_PARALLELISM is set to 1. In this case, identity column values are implicitly assigned in the same order that rows appear in the data file parameter.
- In multi-partition databases, identity column values are assigned in the same order that the rows appear in the data file if the identity column is in the distribution key and if there is a single partitioning agent (that is, if you do not specify multiple partitioning agents or the anyorder file type modifier).

When loading a table in a partitioned database where the table has an identity column in the partitioning key and the identityoverride modifier is not specified, the SAVECOUNT option cannot be specified. When there is an identity column in the partitioning key and identity values are being generated, restarting a load from the load phase on at least one database partition requires restarting the whole load from the beginning of the load phase, which means that there can't be any consistency points.

Note: A load RESTART operation is not permitted if all of the following criteria are met:

- The table being loaded is in a partitioned database environment, and it contains at least one identity column that is either in the distribution key or is referenced by a generated column that is part of the distribution key.
- The identityoverride modifier is not specified.
- The previous load operation that failed included loading database partitions that failed after the load phase.

A load TERMINATE or REPLACE operation should be issued instead.

There are three mutually exclusive ways you can simplify the loading of data into tables that contain an identity column: the identitymissing, the identityignore, and the identityoverride file type modifiers.

Loading data without identity columns

The identitymissing modifier makes loading a table with an identity column more convenient if the input data file does not contain any values (not even NULLS) for the identity column. For example, consider a table defined with the following SQL statement:

```
create table table1 (c1 varchar(30),
                     c2 int generated by default as identity,
                     c3 decimal(7,2),
                     c4 char(1))
```

If you want to load TABLE1 with data from a file (load.del) that has been exported from a table that does not have an identity column, see the following example:

```
Robert, 45.2, J
Mike, 76.9, K
Leo, 23.4, I
```

One way to load this file would be to explicitly list the columns to be loaded through the **LOAD** command as follows:

```
db2 load from load.del of del replace into table1 (c1, c3, c4)
```

For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of loading the file is to use the identitymissing file type modifier as follows:

```
db2 load from load.del of del modified by identitymissing
   replace into table1
```

This command would result in the three columns in the data file being loaded into c1, c3, and c4 of TABLE1. A value will be generated for each row in c2.

Loading data with identity columns

The identityignore modifier indicates to the load utility that even though the input data file contains data for the identity column, the data should be ignored, and an identity value should be generated for each row. For example, a user might want to load TABLE1, as defined previously, from a data file (load.del) containing the following data:

```
Robert, 1, 45.2, J
Mike, 2, 76.9, K
Leo, 3, 23.4, I
```

If the user-supplied values of 1, 2, and 3 are not used for the identity column, you can issue the following **LOAD** command:

```
db2 load from load.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The identityignore modifier simplifies the syntax as follows:

```
db2 load from load.del of del modified by identityignore
   replace into table1
```

Loading data with user-supplied values

The identityoverride modifier is used for loading user-supplied values into a table with a GENERATED ALWAYS identity column. This can be quite useful when migrating data from another database system, and the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the DROPPED TABLE RECOVERY option on the **ROLLFORWARD DATABASE** command. When this modifier is used, any rows with no data (or NULL data) for the identity column are rejected (SQL3116W). You should also note that when using this modifier, it is possible to violate the uniqueness property of GENERATED ALWAYS columns.In this situation, perform a load TERMINATE operation, followed by a subsequent load INSERT or REPLACE operation.

Generated column load considerations

You can load data into a table containing (nonidentity) generated columns whether or not the input data has generated column values. The load utility generates the column values.

If no generated column-related file type modifiers are used, the load utility works according to the following rules:

- Values are created for generated columns when the corresponding row of the data file is missing a value for the column or a NULL value is supplied. If a non-NULL value is supplied for a generated column, the row is rejected (SQL3550W).

- If a NULL value is created for a generated column that is not nullable, the entire row of data is rejected (SQL0407N). This could occur if, for example, a non-nullable generated column is defined as the sum of two table columns that include NULL values in the data file.

There are three mutually exclusive ways you can simplify the loading of data into tables that contain a generated column: the `generatedmissing`, the `generatedignore`, and the `generatedoverride` file type modifiers:

Loading data without generated columns

The `generatedmissing` modifier makes loading a table with generated columns more convenient if the input data file does not contain any values (not even NULLS) for all generated columns present in the table. For example, consider a table defined with the following SQL statement:

```
CREATE TABLE table1 (c1 INT,
                     c2 INT,
                     g1 INT GENERATED ALWAYS AS (c1 + c2),
                     g2 INT GENERATED ALWAYS AS (2 * c1),
                     c3 CHAR(1))
```

If you want to load TABLE1 with data from a file (`load.del`) that has been exported from a table that does not have any generated columns, see the following example:

```
1, 5, J
2, 6, K
3, 7, I
```

One way to load this file would be to explicitly list the columns to be loaded through the LOAD command as follows:

```
DB2 LOAD FROM load.del of del REPLACE INTO table1 (c1, c2, c3)
```

For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of loading the file is to use the `generatedmissing` file type modifier as follows:

```
DB2 LOAD FROM load.del of del MODIFIED BY generatedmissing
   REPLACE INTO table1
```

This command will result in the three columns of data file being loaded into c1, c2, and c3 of TABLE1. Due to the `generatedmissing` modifier, values for columns g1 and g2 of TABLE1 will be generated automatically and will not map to any of the data file columns.

Loading data with generated columns

The `generatedignore` modifier indicates to the load utility that even though the input data file contains data for all generated columns present in the target table, the data should be ignored, and the computed values should be loaded into each generated column. For example, if you want to load TABLE1, as defined previously, from a data file (`load.del`) containing the following data:

```
1, 5, 10, 15, J
2, 6, 11, 16, K
3, 7, 12, 17, I
```

The user-supplied, non-NULL values of `10`, `11`, and `12` (for g1), and `15`, `16`, and `17` (for g2) result in the row being rejected (SQL3550W) if no generated-column related file type modifiers are used. To avoid this, the user could issue the following LOAD command:

```
DB2 LOAD FROM load.del of del method P(1, 2, 5)
   REPLACE INTO table1 (c1, c2, c3)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The generatedignore modifier simplifies the syntax as follows:

```
DB2 LOAD FROM load.del of del MODIFIED BY generatedignore
   REPLACE INTO table1
```

This command will result in the columns of data file being loaded into c1 (with the data 1, 2, 3), c2 (with the data 5,6,7), and c3 (with the data J, K, I) of TABLE1. Due to the generatedignore modifier, values for columns g1 and g2 of TABLE1 will be generated automatically and the data file columns (10, 11, 12 and 15, 16, 17) will be ignored.

Loading data with user-supplied values

The generatedoverride modifier is used for loading user-supplied values into a table with generated columns. This can be useful when migrating data from another database system, or when loading a table from data that was recovered using the RECOVER DROPPED TABLE option of the **ROLLFORWARD DATABASE** command. When this modifier is used, any rows with no data (or NULL data) for non-nullable generated columns are rejected (SQL3116W).

When this modifier is used, the table is placed in the Set Integrity Pending state after the load operation. To take the table out of Set Integrity Pending state without verifying the user-supplied values, issue the following command:

```
SET INTEGRITY FOR table-name GENERATED COLUMN IMMEDIATE
  UNCHECKED
```

To take the table out of the Set Integrity Pending state and force verification of the user-supplied values, issue the following command:

```
SET INTEGRITY FOR table-name IMMEDIATE CHECKED
```

If a generated column is in any of the partitioning, dimension, or distribution keys, the generatedoverride modifier is ignored and the load utility generates values as if the generatedignore modifier is specified. This is done to avoid a scenario where a user-supplied generated column value conflicts with its generated column definition, which would place the resulting record in the wrong physical location, such as the wrong data partition, MDC block, or database partition.

Note: The **LOAD** utility does not support generating column values when one of the generated column expressions contains one of the following:

- a user-defined function that is a compiled compound SQL
- a user-defined function that is FENCED

If you attempt to load into such tables the load operation fails. However, you can provide your own values for these types of generated columns by using the generatedoverride file type modifier.

Moving data using the CURSOR file type

By specifying the CURSOR file type when using the **LOAD** command, you can load the results of an SQL query directly into a target table without creating an intermediate exported file.

Additionally, you can load data from another database by referencing a nickname within the SQL query, by using the DATABASE option within the DECLARE CURSOR statement, or by using the sqlu_remotefetch_entry media entry when using the API interface.

There are three approaches for moving data using the CURSOR file type. The first approach uses the Command Line Processor (CLP), the second the API, and the third uses the ADMIN_CMD procedure. The key differences between the CLP and the ADMIN_CMD procedure are outlined in the following table.

Table 107. Differences between the CLP and ADMIN_CMD procedure.

Differences	CLP	ADMIN_CMD_procedure
Syntax	The query statement as well as the source database used by the cursor are defined outside of the LOAD command using a DECLARE CURSOR statement.	The query statement as well as the source database used by the cursor is defined within the LOAD command using the LOAD from (DATABASE database-alias query-statement)
User authorization for accessing a different database	If the data is in a different database than the one you currently connect to, the DATABASE keyword must be used in the DECLARE CURSOR statement. You can specify the user id and password in the same statement as well. If the user id and password are not specified in the DECLARE CURSOR statement, the user id and password explicitly specified for the source database connection are used to access the target database.	If the data is in a different database than the one you are currently connected to, the DATABASE keyword must be used in the LOAD command before the query statement. The user id and password explicitly specified for the source database connection are required to access the target database. You cannot specify a userid or password for the source database. Therefore, if no userid and password were specified when the connection to the target database was made, or the userid and password specified cannot be used to authenticate against the source database, the ADMIN_CMD procedure cannot be used to perform the load.

To execute a LOAD FROM CURSOR operation from the CLP, a cursor must first be declared against an SQL query. Once this is declared, you can issue the **LOAD** command using the declared cursor's name as the *cursorname* and CURSOR as the file type.

For example:

1. Suppose a source and target table both reside in the same database with the following definitions:

   Table ABC.TABLE1 has 3 columns:

   - ONE INT
   - TWO CHAR(10)

- THREE DATE

Table ABC.TABLE2 has 3 columns:

- ONE VARCHAR
- TWO INT
- THREE DATE

Executing the following CLP commands will load all the data from ABC.TABLE1 into ABC.TABLE2:

```
DECLARE mycurs CURSOR FOR SELECT TWO, ONE, THREE FROM abc.table1
   LOAD FROM mycurs OF cursor INSERT INTO abc.table2
```

Note: The preceding example shows how to load from an SQL query through the CLP. However, loading from an SQL query can also be accomplished through the db2Load API. Define the *piSourceList* of the sqlu_media_list structure to use the *sqlu_statement_entry* structure and SQLU_SQL_STMT media type and define the *piFileType* value as SQL_CURSOR.

2. Suppose the source and target tables reside in different databases with the following definitions:

Table ABC.TABLE1 in database 'dbsource' has 3 columns:

- ONE INT
- TWO CHAR(10)
- THREE DATE

Table ABC.TABLE2 in database 'dbtarget' has 3 columns:

- ONE VARCHAR
- TWO INT
- THREE DATE

Provided that you have enabled federation and cataloged the data source ('dsdbsource'), you can declare a nickname against the source database, then declare a cursor against this nickname, and invoke the LOAD command with the FROM CURSOR option, as demonstrated in the following example:

```
CREATE NICKNAME myschema1.table1 FOR dsdbsource.abc.table1
DECLARE mycurs CURSOR FOR SELECT TWO,ONE,THREE FROM myschema1.table1
LOAD FROM mycurs OF cursor INSERT INTO abc.table2
```

Or, you can use the DATABASE option of the DECLARE CURSOR statement, as demonstrated in the following example:

```
DECLARE mycurs CURSOR DATABASE dbsource USER dsciaraf USING mypasswd
FOR SELECT TWO,ONE,THREE FROM abc.table1
LOAD FROM mycurs OF cursor INSERT INTO abc.table2
```

Using the DATABASE option of the DECLARE CURSOR statement (also known as the remotefetch media type when using the Load API) has some benefits over the nickname approach:

Performance

Fetching of data using the remotefetch media type is tightly integrated within a load operation. There are fewer layers of transition to fetch a record compared to the nickname approach. Additionally, when source and target tables are distributed identically in a multi-partition database, the load utility can parallelize the fetching of data, which can further improve performance.

Ease of use

> There is no need to enable federation, define a remote datasource, or declare a nickname. Specifying the DATABASE option (and the USER and USING options if necessary) is all that is required.

While this method can be used with cataloged databases, the use of nicknames provides a robust facility for fetching from various data sources which cannot simply be cataloged.

To support this remotefetch functionality, the load utility makes use of infrastructure which supports the SOURCEUSEREXIT facility. The load utility spawns a process which executes as an application to manage the connection to the source database and perform the fetch. This application is associated with its own transaction and is not associated with the transaction under which the load utility is running.

Note:

1. The previous example shows how to load from an SQL query against a cataloged database through the CLP using the DATABASE option of the DECLARE CURSOR statement. However, loading from an SQL query against a cataloged database can also be done through the db2Load API, by defining the *piSourceList* and *piFileTypevalues* of the *db2LoadStruct* structure to use the sqlu_remotefetch_entry media entry and SQLU_REMOTEFETCH media type respectively.
2. As demonstrated in the previous example, the source column types of the SQL query do not need to be identical to their target column types, although they do have to be compatible.

Restrictions

When loading from a cursor defined using the DATABASE option (or equivalently when using the sqlu_remotefetch_entry media entry with the db2Load API), the following restrictions apply:

1. The SOURCEUSEREXIT option cannot be specified concurrently.
2. The METHOD N option is not supported.
3. The usedefaults file type modifier is not supported.

Refreshing dependent immediate materialized query tables

If the underlying table of an immediate refresh materialized query table is loaded using the INSERT option, executing the SET INTEGRITY statement on the dependent materialized query tables defined with REFRESH IMMEDIATE results in an incremental refresh of the materialized query table.

During an incremental refresh, the rows corresponding to the appended rows in the underlying tables are updated and inserted into the materialized query tables. Incremental refresh is faster in the case of large underlying tables with small amounts of appended data. There are cases in which incremental refresh is not allowed, and full refresh (that is, recomputation of the materialized query table definition query) is used.

When the INCREMENTAL option is specified, but incremental processing of the materialized query table is not possible, an error is returned if:

- A load replace operation has taken place into an underlying table of the materialized query table or the NOT LOGGED INITIALLY WITH EMPTY TABLE option has been activated since the last integrity check on the underlying table.
- The materialized query table has been loaded (in either REPLACE or INSERT mode).
- An underlying table has been taken out of Set Integrity Pending state before the materialized query table is refreshed by using the `FULL ACCESS` option during integrity checking.
- An underlying table of the materialized query table has been checked for integrity non-incrementally.
- The materialized query table was in Set Integrity Pending state before an upgrade.
- The table space containing the materialized query table or its underlying table has been rolled forward to a point in time, and the materialized query table and its underlying table reside in different table spaces.

If the materialized query table has one or more `W` values in the CONST_CHECKED column of the SYSCAT.TABLES catalog, and if the `NOT INCREMENTAL` option is not specified in the SET INTEGRITY statement, the table is incrementally refreshed and the CONST_CHECKED column of SYSCAT.TABLES is marked `U` to indicate that not all data has been verified by the system.

The following example illustrates a load insert operation into the underlying table UTI of the materialized query table AST1. UT1 is checked for data integrity and is placed in the no data movement mode. UT1 is put back into full access state once the incremental refresh of AST1 is complete. In this scenario, both the integrity checking for UT1 and the refreshing of AST1 are processed incrementally.

```
LOAD FROM IMTFILE1.IXF of IXF INSERT INTO UT1;
LOAD FROM IMTFILE2.IXF of IXF INSERT INTO UT1;
SET INTEGRITY FOR UT1 IMMEDIATE CHECKED;
REFRESH TABLE AST1;
```

MDC and ITC load considerations

The following restrictions apply when loading data into multidimensional clustering (MDC) and insert time clustering (ITC) tables:

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

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

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

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

Performance Considerations

To improve the performance of the load utility when loading MDC tables with more than one dimension, the *util_heap_sz* database configuration parameter value should be increased. The mdc-load algorithm performs significantly better when more memory is available to the utility. This reduces disk I/O during the clustering of data that is performed during the load phase. Beginning in version 9.5, the value of the DATA BUFFER option of the **LOAD** command can temporarily exceed *util_heap_sz* if more memory is available in the system. .

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

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

A system temporary table with an index is used to load data into MDC and ITC tables. The size of the table is proportional to the number of distinct cells loaded. The size of each row in the table is proportional to the size of the MDC dimension key. ITC tables only have one cell and use a 2-byte dimension key. To minimize disk I/O caused by the manipulation of this table during a load operation, ensure that the buffer pool for the temporary table space is large enough.

Partitioned tables load considerations

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

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

General load behavior

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

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

Handling of invalid rows

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

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

History file

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

Terminating a load operation

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

Generated columns

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

Data availability

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

Important: Starting with Version 10.1 Fix Pack 1, the ALLOW READ ACCESS parameter is deprecated and might be removed in a future release. For more details, see “ALLOW READ ACCESS parameter in the LOAD command is deprecated” in *What's New for DB2 Version 10.1*.

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

Data partition states

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

Error isolation

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

Other considerations

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

Loading XML data

The load utility can be used for the efficient movement of large volumes of XML data into tables.

When loading data into an XML table column, you can use the XML FROM option to specify the paths of the input XML data file or files. For example, to load data from an XML file `/home/user/xmlpath/xmlfile1.xml` you could use the following command:

```
   LOAD FROM data1.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

The delimited ASCII input file `data1.del` contains an XML data specifier (XDS) that describes the location of the XML data to load. For example, the following XDS describes an XML document at offset 123 bytes in file `xmldata.ext` that is 456 bytes in length:

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' />
```

Loading XML data using a declared cursor is supported. The following example declares a cursor and uses the cursor and the **LOAD** command to add data from the table CUSTOMERS into the table LEVEL1_CUSTOMERS:

```
DECLARE cursor_income_level1 CURSOR FOR
   SELECT * FROM customers
   WHERE XMLEXISTS('$DOC/customer[income_level=1]');

LOAD FROM cursor_income_level1 OF CURSOR INSERT INTO level1_customers;
```

The `ANYORDER` file type modifier of the **LOAD** command is supported for loading XML data into an XML column.

During load, distribution statistics are not collected for columns of type XML.

Loading XML data in a partitioned database environment

For tables that are distributed among database partitions, you can load XML data from XML data files into the tables in parallel. When loading XML data from files into tables, the XML data files must be read-accessible to all the database partitions where loading is taking place

Validating inserted documents against schemas

The XMLVALIDATE option allows XML documents to be validated against XML schemas as they are loaded. In the following example, incoming XML documents are validated against the schema identified by the XDS in the delimited ASCII input file `data2.del`:

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T2
```

In this case, the XDS contains an SCH attribute with the fully qualified SQL identifier of the XML schema to use for validation, "S1.SCHEMA_A":

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' SCH='S1.SCHEMA_A' />
```

Specifying parse options

You can use the XMLPARSE option to specify whether whitespace in the loaded XML documents is preserved or stripped. In the following example, all loaded XML documents are validated against the schema with SQL identifier "S2.SCHEMA_A" and these documents are parsed with whitespace preserved:

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING SCHEMA S2.SCHEMA_A INSERT INTO USER.T1
```

Load in partitioned database environments

In a multi-partition database, large amounts of data are located across many database partitions. Distribution keys are used to determine on which database partition each portion of the data resides. The data must be *distributed* before it can be loaded at the correct database partition.

When loading tables in a multi-partition database, the load utility can:

- Distribute input data in parallel
- Load data simultaneously on corresponding database partitions
- Transfer data from one system to another system

Loading data into a multi-partition database takes place in two phases: the *setup phase*, during which database partition resources such as table locks are acquired, and the *load phase*, during which the data is loaded into the database partitions. You can use the `ISOLATE_PART_ERRS` option of the **`LOAD`** command to select how errors are handled during either of these phases, and how errors on one or more of the database partitions affect the load operation on the database partitions that are not experiencing errors.

When loading data into a multi-partition database you can use one of the following modes:

PARTITION_AND_LOAD
Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

PARTITION_ONLY
Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. Each file includes a

partition header that specifies how the data was distributed across the database partitions, and that the file can be loaded into the database using the LOAD_ONLY mode.

LOAD_ONLY
Data is assumed to be already distributed across the database partitions; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions.

LOAD_ONLY_VERIFY_PART
Data is assumed to be already distributed across the database partitions, but the data file does not contain a partition header. The distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dump file if the `dumpfile` file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning is written to the load message file for that database partition.

ANALYZE
An optimal distribution map with even distribution across all database partitions is generated.

Concepts and terminology

The following terminology is used when discussing the behavior and operation of the load utility in a partitioned database environment with multiple database partitions:

- The *coordinator partition* is the database partition to which the user connects in order to perform the load operation. In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, it is assumed that the data file resides on this database partition unless the `CLIENT` option of the **`LOAD`** command is specified. Specifying `CLIENT` indicates that the data to be loaded resides on a remotely connected client.
- In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, the *pre-partitioning agent* reads the user data and distributes it in round-robin fashion to the *partitioning agents* which then distribute the data. This process is always performed on the coordinator partition. A maximum of one partitioning agent is allowed per database partition for any load operation.
- In the PARTITION_AND_LOAD, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, *load agents* run on each output database partition and coordinate the loading of data to that database partition.
- *Load to file agents* run on each output database partition during a PARTITION_ONLY load operation. They receive data from partitioning agents and write it to a file on their database partition.
- The `SOURCEUSEREXIT` option provides a facility through which the load utility can execute a customized script or executable, referred to herein as the *user exit*.

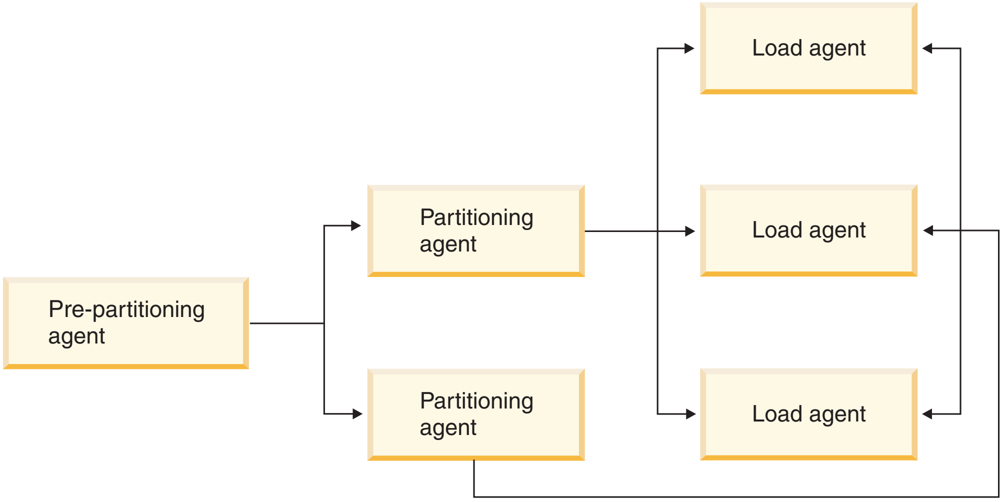


Figure 46. Partitioned Database Load Overview. The source data is read by the pre-partitioning agent, and approximately half of the data is sent to each of two partitioning agents which distribute the data and send it to one of three database partitions. The load agent at each database partition loads the data.

Loading data in a partitioned database environment

Using the load utility to load data into a partitioned database environment.

Before you begin

Before loading a table in a multi-partition database:

- Ensure that the **svcename** database manager configuration parameter and the **DB2COMM** profile registry variable are set correctly. This step is important because the load utility uses TCP/IP to transfer data from the pre-partitioning agent to the partitioning agents, and from the partitioning agents to the loading database partitions.
- Before invoking the load utility, you must be connected to (or be able to implicitly connect to) the database into which you want to load the data.
- Since the load utility issues a COMMIT statement, complete all transactions and release any locks by issuing either a COMMIT or a ROLLBACK statement before beginning the load operation. If the PARTITION_AND_LOAD, PARTITION_ONLY, or ANALYZE mode is being used, the data file that is being loaded must reside on this database partition unless:
  1. The **CLIENT** parameter has been specified, in which case the data must reside on the client machine;
  2. The input source type is CURSOR, in which case there is no input file.
- Run the Design Advisor to determine the best database partition for each table. For more information, see “The Design Advisor” in *Troubleshooting and Tuning Database Performance*.

Restrictions

The following restrictions apply when using the load utility to load data in a multi-partition database:

- The location of the input files to the load operation cannot be a tape device.

- The **ROWCOUNT** parameter is not supported unless the ANALYZE mode is being used.
- If the target table has an identity column that is needed for distributing and the **identityoverride** file type modifier is not specified, or if you are using multiple database partitions to distribute and then load the data, the use of a **SAVECOUNT** greater than 0 on the **LOAD** command is not supported.
- If an identity column forms part of the distribution key, only the PARTITION_AND_LOAD mode is supported.
- The LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes cannot be used with the **CLIENT** parameter of the **LOAD** command.
- The LOAD_ONLY_VERIFY_PART mode cannot be used with the CURSOR input source type.
- The distribution error isolation modes LOAD_ERRS_ONLY and SETUP_AND_LOAD_ERRS cannot be used with the **ALLOW READ ACCESS** and **COPY YES** parameters of the **LOAD** command.
- Multiple load operations can load data into the same table concurrently if the database partitions specified by the**OUTPUT_DBPARTNUMS** and **PARTITIONING_DBPARTNUMS** options do not overlap. For example, if a table is defined on database partitions 0 through 3, one load operation can load data into database partitions 0 and 1 while a second load operation can load data into database partitions 2 and 3. If the database partitions specified by the **PARTITIONING_DBPARTNUMS** options do overlap, then load will automatically choose a **PARTITIONING_DBPARTNUMS** parameter where no load partitioning subagent is already executing on the table, or fail if none are available.

  Starting with Version 9.7 Fix Pack 6, if the database partitions specified by the **PARTITIONING_DBPARTNUMS** options do overlap, the load utility automatically tries to pick up a **PARTITIONING_DBPARTNUMS** parameter from the database partitions indicated by **OUTPUT_DBPARTNUMS** where no load partitioning subagent is already executing on the table, or fail if none are available.

  It is strongly recommended that if you are going to explicitly specify partitions with the **PARTITIONING_DBPARTNUMS** option, you should use that option with all concurrent **LOAD** commands, with each command specifying different partitions. If you only specify **PARTITIONING_DBPARTNUMS** on some of the concurrent load commands or if you specify overlapping partitions, the **LOAD** command will need to pick alternate partitioning nodes for at least some of the concurrent loads, and in rare cases the command might fail (SQL2038N).
- Only non-delimited ASCII (ASC) and Delimited ASCII (DEL) files can be distributed across tables spanning multiple database partitions. PC/IXF files cannot be distributed, however, you can load a PC/IXF file into a table that is distributed over multiple database partitions by using the load operation in the LOAD_ONLY_VERIFY_PART mode.

Example

The following examples illustrate how to use the **LOAD** command to initiate various types of load operations. The database used in the following examples has five database partitions: 0, 1, 2, 3 and 4. Each database partition has a local directory /db2/data/. Two tables, TABLE1 and TABLE2, are defined on database partitions 0, 1, 3 and 4. When loading from a client, the user has access to a remote client that is not one of the database partitions.

Distribute and load example

In this scenario, you are connected to a database partition that might or might not be a database partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this database partition. To load the data from `load.del` into all of the database partitions where TABLE1 is defined, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
```

Note: In this example, default values are used for all of the configuration parameters for partitioned database environments: The **MODE** parameter defaults to PARTITION_AND_LOAD. The **OUTPUT_DBPARTNUMS** parameter defaults to all database partitions on which TABLE1 is defined. The **PARTITIONING_DBPARTNUMS** defaults to the set of database partitions selected according to the **LOAD** command rules for choosing database partitions when none are specified.

To perform a load operation where data is distributed over database partitions 3 and 4, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
 PARTITIONED DB CONFIG PARTITIONING_DBPARTNUMS (3,4)
```

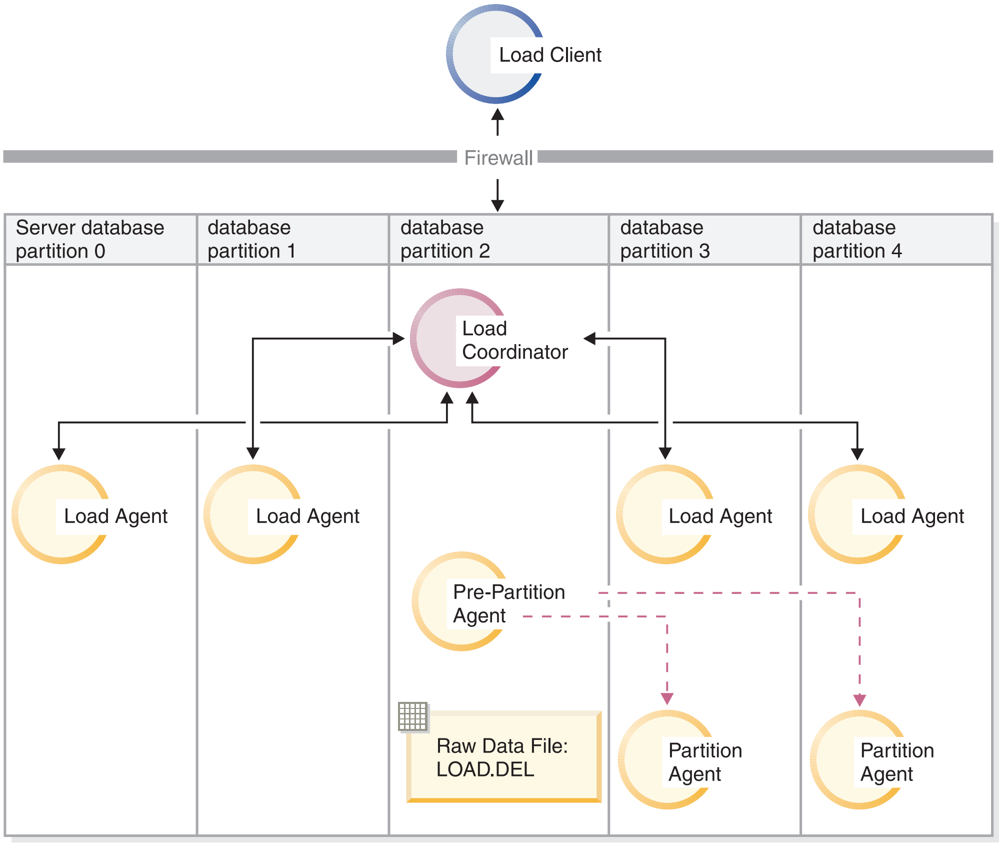


Figure 47. Loading data into database partitions 3 and 4.. This diagram illustrates the behavior resulting when the previous command is issued. Data is loaded into database partitions 3 and 4.

Distribute only example

In this scenario, you are connected to a database partition that might or might not be a database partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this database partition.

To distribute (but not load) `load.del` to all the database partitions on which TABLE1 is defined, using database partitions 3 and 4 issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE PARTITION_ONLY
 PART_FILE_LOCATION /db2/data
 PARTITIONING_DBPARTNUMS (3,4)
```

This results in a file `load.del.`*`xxx`* being stored in the `/db2/data` directory on each database partition, where *xxx* is a three-digit representation of the database partition number.

To distribute the `load.del` file to database partitions 1 and 3, using only one partitioning agent running on database partition 0 (which is the default for **`PARTITIONING_DBPARTNUMS`**), issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE PARTITION_ONLY
 PART_FILE_LOCATION /db2/data
 OUTPUT_DBPARTNUMS (1,3)
```

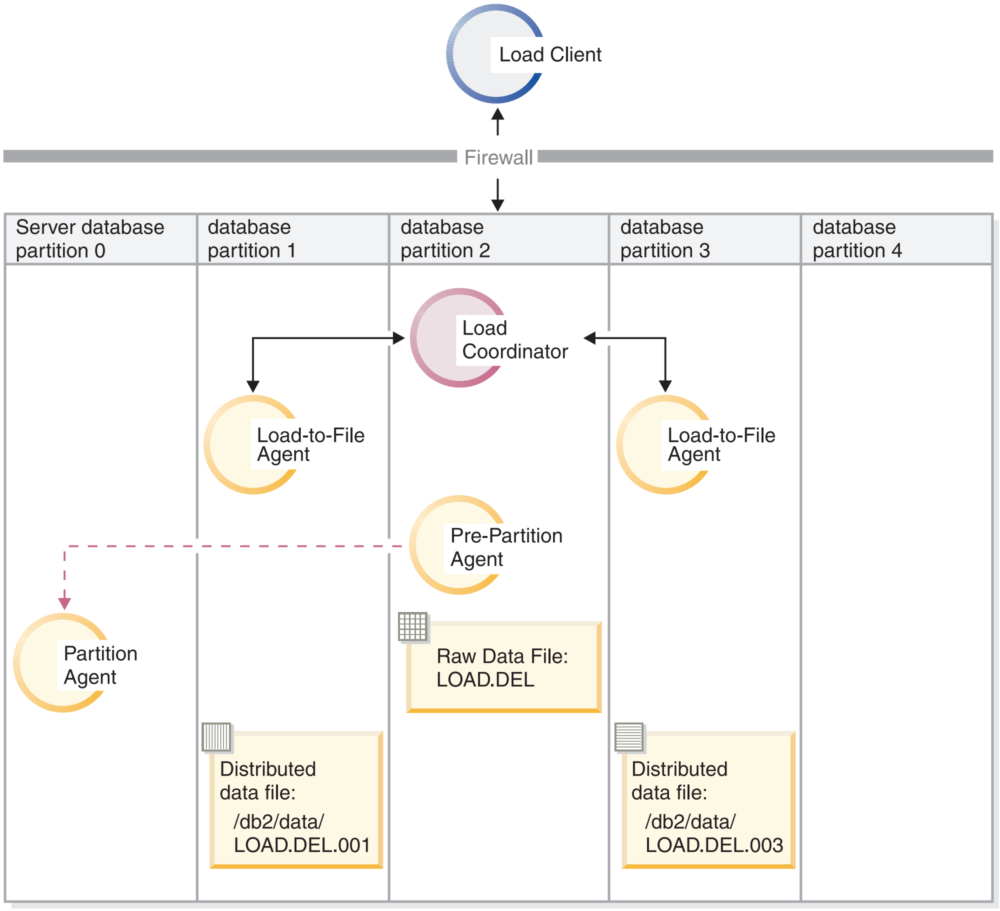


Figure 48. Loading data into database partitions 1 and 3 using one partitioning agent.. This diagram illustrates the behavior that results when the previous command is issued. Data is loaded into database partitions 1 and 3, using one partitioning agent running on database partition 0.

Load only example

If you have already performed a load operation in the PARTITION_ONLY mode and want to load the partitioned files in the /db2/data directory of each loading database partition to all the database partitions on which TABLE1 is defined, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE LOAD_ONLY
 PART_FILE_LOCATION /db2/data
```

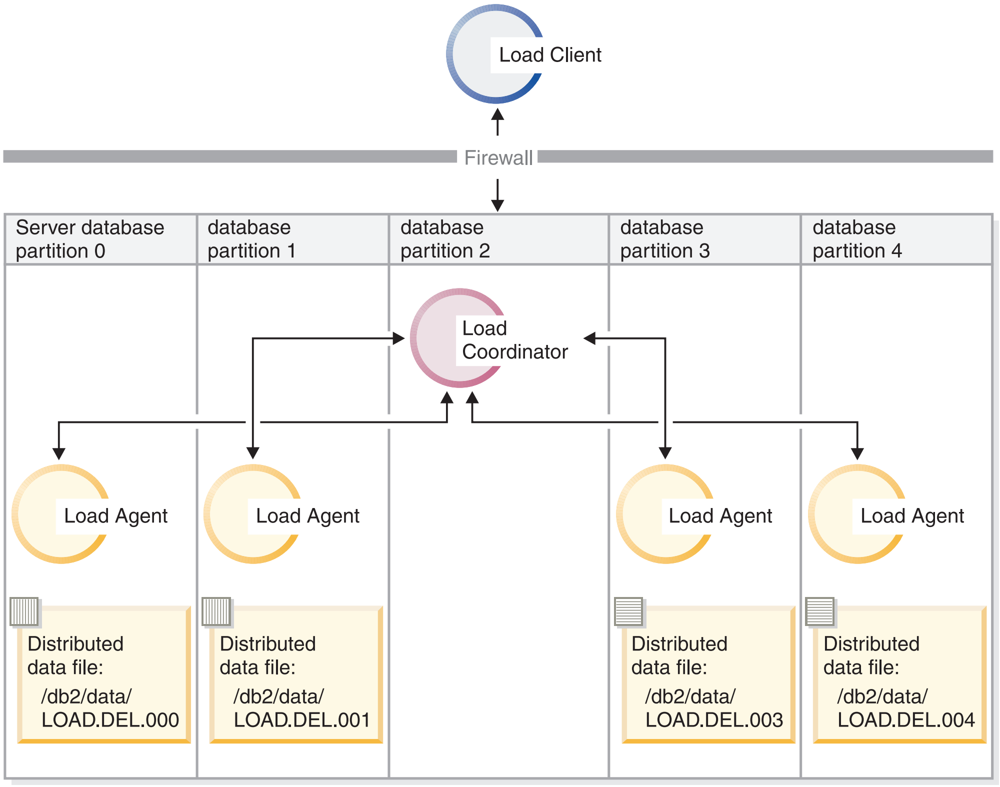

Load Client
Firewall
Server database partition 0
database partition 1
database partition 2
database partition 3
database partition 4
Load Coordinator
Load Agent
Load Agent
Load Agent
Load Agent
Distributed data file: /db2/data/ LOAD.DEL.000
Distributed data file: /db2/data/ LOAD.DEL.001
Distributed data file: /db2/data/ LOAD.DEL.003
Distributed data file: /db2/data/ LOAD.DEL.004

Figure 49. Loading data into all database partitions where a specific table is defined.. This diagram illustrates the behavior resulting when the previous command is issued. Distributed data is loaded to all database partitions where TABLE1 is defined.

To load into database partition 4 only, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE LOAD_ONLY
 PART_FILE_LOCATION /db2/data
 OUTPUT_DBPARTNUMS (4)
```

Loading pre-distributed files without distribution map headers

The **LOAD** command can be used to load data files without distribution headers directly into several database partitions. If the data files exist in the /db2/data directory on each database partition where TABLE1 is defined and have the name load.del.*xxx*, where *xxx* is the database partition number, the files can be loaded by issuing the following command:

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
 REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
 PART_FILE_LOCATION /db2/data
```

To load the data into database partition 1 only, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
 REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
 PART_FILE_LOCATION /db2/data
 OUTPUT_DBPARTNUMS (1)
```

Note: Rows that do not belong on the database partition from which they were loaded are rejected and put into the dump file, if one has been specified.

Loading from a remote client to a multi-partition database

To load data into a multi-partition database from a file that is on a remote client, you must specify the **CLIENT** parameter of the **LOAD** command. This parameter indicates that the data file is not on a server partition. For example:

```
LOAD CLIENT FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

Note: You cannot use the LOAD_ONLY or LOAD_ONLY_VERIFY_PART modes with the **CLIENT** parameter.

Loading from a cursor

As in a single-partition database, you can load from a cursor into a multi-partition database. In this example, for the PARTITION_ONLY and LOAD_ONLY modes, the **PART_FILE_LOCATION** parameter must specify a fully qualified file name. This name is the fully qualified base file name of the distributed files that are created or loaded on each output database partition. Multiple files can be created with the specified base name if there are LOB columns in the target table.

To distribute all the rows in the answer set of the statement SELECT * FROM TABLE1 to a file on each database partition named /db2/data/select.out.*xxx* (where *xxx* is the database partition number), for future loading into TABLE2, issue the following commands:

```
DECLARE C1 CURSOR FOR SELECT * FROM TABLE1

LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
 PARTITIONED DB CONFIG MODE PARTITION_ONLY
 PART_FILE_LOCATION /db2/data/select.out
```

The data files produced by the previous operation can then be loaded by issuing the following **LOAD** command:

```
LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
 PARTITIONED CB CONFIG MODE LOAD_ONLY
 PART_FILE_LOCATION /db2/data/select.out
```

Load sessions in a partitioned database environment - CLP examples

The following examples demonstrate loading data in a multi-partition database.

The database has four database partitions numbered 0 through 3. Database WSDB is defined on all of the database partitions, and table TABLE1 resides in the default database partition group which is also defined on all of the database partitions.

Example 1

To load data into TABLE1 from the user data file load.del which resides on database partition 0, connect to database partition 0 and then issue the following command:

```
load from load.del of del replace into table1
```

If the load operation is successful, the output will be as follows:

```
Agent Type     Node     SQL Code     Result
_______________________________________________
LOAD           000      +00000000    Success.
_______________________________________________
LOAD           001      +00000000    Success.
_______________________________________________
LOAD           002      +00000000    Success.
_______________________________________________
LOAD           003      +00000000    Success.
_______________________________________________
PARTITION      001      +00000000    Success.
_______________________________________________
PRE_PARTITION  000      +00000000    Success.
_______________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
_______________________________________________

Summary of Partitioning Agents:
Rows Read                   = 100000
Rows Rejected               = 0
Rows Partitioned            = 100000

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

The output indicates that there was one load agent on each database partition and each ran successfully. It also shows that there was one pre-partitioning agent running on the coordinator partition and one partitioning agent running on database partition 1. These processes completed successfully with a normal SQL return code of 0. The statistical summary shows that the pre-partitioning agent read 100,000 rows, the partitioning agent distributed 100,000 rows, and the sum of all rows loaded by the load agents is 100,000.

Example 2

In the following example, data is loaded into TABLE1 in the PARTITION_ONLY mode. The distributed output files is stored on each of the output database partitions in the directory /db/data:

```
load from load.del of del replace into table1 partitioned db config mode
   partition_only part_file_location /db/data
```

The output from the load command is as follows:

```
Agent Type     Node     SQL Code     Result
_______________________________________________
LOAD_TO_FILE   000      +00000000    Success.
_______________________________________________
LOAD_TO_FILE   001      +00000000    Success.
_______________________________________________
```

```
LOAD_TO_FILE   002      +00000000    Success.
___________________________________________________
LOAD_TO_FILE   003      +00000000    Success.
___________________________________________________
PARTITION      001      +00000000    Success.
___________________________________________________
PRE_PARTITION  000      +00000000    Success.
___________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 100000
Rows Rejected               = 0
Rows Partitioned            = 100000
```

The output indicates that there was a load-to-file agent running on each output database partition, and these agents ran successfully. There was a pre-partitioning agent on the coordinator partition, and a partitioning agent running on database partition 1. The statistical summary indicates that 100,000 rows were successfully read by the pre-partitioning agent and 100,000 rows were successfully distributed by the partitioning agent. Since no rows were loaded into the table, no summary of the number of rows loaded appears.

Example 3

To load the files that were generated during the PARTITION_ONLY load operation shown previously, issue the following command:

```
load from load.del of del replace into table1 partitioned db config mode
    load_only part_file_location /db/data
```

The output from the load command will be as follows:

```
Agent Type     Node     SQL Code     Result
___________________________________________________
LOAD           000      +00000000    Success.
___________________________________________________
LOAD           001      +00000000    Success.
___________________________________________________
LOAD           002      +00000000    Success.
___________________________________________________
LOAD           003      +00000000    Success.
___________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
___________________________________________________

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

The output indicates that the load agents on each output database partition ran successfully and that the sum of the number of rows loaded by all load agents is 100,000. No summary of rows distributed is indicated since distribution was not performed.

Example 4

If the following LOAD command is issued:

```
load from load.del of del replace into table1
```

and one of the loading database partitions runs out of space in the table space during the load operation, the following output might be returned:

```
SQL0289N  Unable to allocate new pages in table space "DMS4KT".
SQLSTATE=57011

Agent Type     Node     SQL Code     Result
________________________________________________________________
LOAD           000      +00000000    Success.
________________________________________________________________
LOAD           001      -00000289    Error. May require RESTART.
________________________________________________________________
LOAD           002      +00000000    Success.
________________________________________________________________
LOAD           003      +00000000    Success.
________________________________________________________________
PARTITION      001      +00000000    Success.
________________________________________________________________
PRE_PARTITION  000      +00000000    Success.
________________________________________________________________
RESULTS:       3 of 4 LOADs completed successfully.
________________________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 0
Rows Rejected               = 0
Rows Partitioned            = 0

Summary of LOAD Agents:
Number of rows read         = 0
Number of rows skipped      = 0
Number of rows loaded       = 0
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 0
```

The output indicates that the load operation returned error SQL0289. The database partition summary indicates that database partition 1 ran out of space. If additional space is added to the containers of the table space on database partition 1, the load operation can be restarted as follows:

```
load from load.del of del restart into table1
```

Load features for maintaining referential integrity

Although the load utility is typically more efficient than the import utility, it requires a number of features to ensure the referential integrity of the information being loaded:

- **Table locks**, which provide concurrency control and prevent uncontrolled data access during a load operation
- **Table states** and **table space states**, which can either control access to data or elicit specific user actions
- **Load exception tables**, which ensure that rows of invalid data are not simply deleted without your knowledge

Checking for integrity violations following a load operation

Following a load operation, the loaded table might be in set integrity pending state in either READ or NO ACCESS mode if any of the following conditions exist:

- The table has table check constraints or referential integrity constraints defined on it.

- The table has generated columns and a V7 or earlier client was used to initiate the load operation.
- The table has descendent immediate materialized query tables or descendent immediate staging tables referencing it.
- The table is a staging table or a materialized query table.

The STATUS flag of the SYSCAT.TABLES entry corresponding to the loaded table indicates the set integrity pending state of the table. For the loaded table to be fully usable, the STATUS must have a value of N and the ACCESS MODE must have a value of F, indicating that the table is fully accessible and in normal state.

If the loaded table has descendent tables, the SET INTEGRITY PENDING CASCADE parameter can be specified to indicate whether or not the set integrity pending state of the loaded table should be immediately cascaded to the descendent tables.

If the loaded table has constraints as well as descendent foreign key tables, dependent materialized query tables and dependent staging tables, and if all of the tables are in normal state before the load operation, the following will result based on the load parameters specified:

INSERT, ALLOW READ ACCESS, and SET INTEGRITY PENDING CASCADE IMMEDIATE
The loaded table, its dependent materialized query tables and dependent staging tables are placed in set integrity pending state with read access.

INSERT, ALLOW READ ACCESS, and SET INTEGRITY PENDING CASCADE DEFERRED
Only the loaded table is placed in set integrity pending with read access. Descendent foreign key tables, descendent materialized query tables and descendent staging tables remain in their original states.

INSERT, ALLOW NO ACCESS, and SET INTEGRITY PENDING CASCADE IMMEDIATE
The loaded table, its dependent materialized query tables and dependent staging tables are placed in set integrity pending state with no access.

INSERT or REPLACE, ALLOW NO ACCESS, and SET INTEGRITY PENDING CASCADE DEFERRED
Only the loaded table is placed in set integrity pending state with no access. Descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables remain in their original states.

REPLACE, ALLOW NO ACCESS, and SET INTEGRITY PENDING CASCADE IMMEDIATE
The table and all its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables are placed in set integrity pending state with no access.

Note: Specifying the ALLOW READ ACCESS option in a load replace operation results in an error.

To remove the set integrity pending state, use the SET INTEGRITY statement. The SET INTEGRITY statement checks a table for constraints violations, and takes the table out of set integrity pending state. If all the load operations are performed in

INSERT mode, the SET INTEGRITY statement can be used to incrementally process the constraints (that is, it checks only the appended portion of the table for constraints violations). For example:

```
db2 load from infile1.ixf of ixf insert into table1
db2 set integrity for table1 immediate checked
```

Only the appended portion of TABLE1 is checked for constraint violations. Checking only the appended portion for constraints violations is faster than checking the entire table, especially in the case of a large table with small amounts of appended data.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for setting integrity. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

If a table is loaded with the SET INTEGRITY PENDING CASCADE DEFERRED option specified, and the SET INTEGRITY statement is used to check for integrity violations, the descendent tables are placed in set integrity pending state with no access. To take the tables out of this state, you must issue an explicit request.

If a table with dependent materialized query tables or dependent staging tables is loaded using the INSERT option, and the SET INTEGRITY statement is used to check for integrity violations, the table is taken out of set integrity pending state and placed in No Data Movement state. This is done to facilitate the subsequent incremental refreshes of the dependent materialized query tables and the incremental propagation of the dependent staging tables. In the No Data Movement state, operations that might cause the movement of rows within the table are not allowed.

You can override the No Data Movement state by specifying the FULL ACCESS option when you issue the SET INTEGRITY statement. The table is fully accessible, however a full re-computation of the dependent materialized query tables takes place in subsequent REFRESH TABLE statements and the dependent staging tables are forced into an incomplete state.

If the ALLOW READ ACCESS option is specified for a load operation, the table remains in read access state until the SET INTEGRITY statement is used to check for constraints violations. Applications can query the table for data that existed before the load operation once it has been committed, but will not be able to view the newly loaded data until the SET INTEGRITY statement is issued.

Several load operations can take place on a table before checking for constraints violations. If all of the load operations are completed in ALLOW READ ACCESS mode, only the data that existed in the table before the first load operation is available for queries.

One or more tables can be checked in a single invocation of this statement. If a dependent table is to be checked on its own, the parent table can not be in set integrity pending state. Otherwise, both the parent table and the dependent table must be checked at the same time. In the case of a referential integrity cycle, all the tables involved in the cycle must be included in a single invocation of the SET INTEGRITY statement. It might be convenient to check the parent table for constraints violations while a dependent table is being loaded. This can only occur if the two tables are not in the same table space.

When issuing the SET INTEGRITY statement, you can specify the INCREMENTAL option to explicitly request incremental processing. In most cases, this option is not needed, because the DB2 database selects incremental processing. If incremental processing is not possible, full processing is used automatically. When the INCREMENTAL option is specified, but incremental processing is not possible, an error is returned if:

- New constraints are added to the table while it is in set integrity pending state.
- A load replace operation takes place, or the NOT LOGGED INITIALLY WITH EMPTY TABLE option is activated, after the last integrity check on the table.
- A parent table is load replaced or checked for integrity non-incrementally.
- The table is in set integrity pending state before an upgrade. Full processing is required the first time the table is checked for integrity after an upgrade.
- The table space containing the table or its parent is rolled forward to a point in time and the table and its parent reside in different table spaces.

If a table has one or more `W` values in the CONST_CHECKED column of the SYSCAT.TABLES catalog, and if the NOT INCREMENTAL option is not specified in the SET INTEGRITY statement, the table is incrementally processed and the CONST_CHECKED column of SYSCAT.TABLES is marked as `U` to indicate that not all data has been verified by the system.

The SET INTEGRITY statement does not activate any DELETE triggers as a result of deleting rows that violate constraints, but once the table is removed from set integrity pending state, triggers are active. Thus, if you correct data and insert rows from the exception table into the loaded table, any INSERT triggers defined on the table are activated. The implications of this should be considered. One option is to drop the INSERT trigger, insert rows from the exception table, and then re-create the INSERT trigger.

Table locking during load operations

In most cases, the load utility uses table level locking to restrict access to tables. The level of locking depends on the stage of the load operation and whether it was specified to allow read access.

A load operation in `ALLOW NO ACCESS` mode uses a super exclusive lock (Z-lock) on the table for the duration of the load.

Before a load operation in `ALLOW READ ACCESS` mode begins, the load utility waits for all applications that began before the load operation to release their locks on the target table. At the beginning of the load operation, the load utility acquires an update lock (U-lock) on the table. It holds this lock until the data is being committed. When the load utility acquires the U-lock on the table, it waits for all applications that hold locks on the table before the start of the load operation to release them, even if they have compatible locks. This is achieved by temporarily upgrading the U-lock to a Z-lock which does not conflict with new table lock requests on the target table as long as the requested locks are compatible with the load operation's U-lock. When data is being committed, the load utility upgrades the lock to a Z-lock, so there can be some delay in commit time while the load utility waits for applications with conflicting locks to finish.

Note: The load operation can time out while it waits for the applications to release their locks on the table before loading. However, the load operation does not time out while waiting for the Z-lock needed to commit the data.

Applications with conflicting locks

Use the LOCK WITH FORCE option of the **LOAD** command to force off applications holding conflicting locks on a target table so that the load operation can proceed. Before a load operation in ALLOW READ ACCESS mode can proceed, applications holding the following locks are forced off:

- Table locks that conflict with a table update lock (for example, import or insert).
- All table locks that exist at the commit phase of the load operation.

Applications holding conflicting locks on the system catalog tables are not forced off by the load utility. If an application is forced off the system by the load utility, the application loses its database connection, and an error is returned (SQL1224N).

When you specify the COPY NO option for a load operation on a recoverable database, all objects in the target table space are locked in share mode before the table space is placed in the Backup Pending state. This occurs regardless of the access mode. If you specify the LOCK WITH FORCE option, all applications holding locks on objects in the table space that conflict with a share lock are forced off.

Table space states during and after load operations

The load utility uses table space states to preserve database consistency during a load operation. These states work by controlling access to data or eliciting user actions.

The load utility does not quiesce (put persistent locks on) the table spaces involved in the load operation and uses table space states only for load operations for which you specify the **COPY NO** parameter.

You can check table space states by using the **LIST TABLESPACES** command. Table spaces can be in multiple states simultaneously. The states returned by **LIST TABLESPACES** are as follows:

Normal

The Normal state is the initial state of a table space after it is created, indicating that no (abnormal) states currently affect it.

Load in Progress

The Load in Progress state indicates that there is a load in progress on the table space. This state prevents the backup of dependent tables during the load. The table space state is distinct from the Load in Progress table state (which is used in all load operations) because the load utility places table spaces in the Load in Progress state only when you specify the **COPY NO** parameter for a recoverable database. The table spaces remain in this state for the duration of the load operation.

Backup Pending

If you perform a load operation for a recoverable database and specify the **COPY NO** parameter, table spaces are placed in the Backup Pending table space state after the first commit. You cannot update a table space in the Backup Pending state. You can remove the table space from the Backup Pending state only by backing up the table space. Even if you cancel the load operation, the table space remains in the Backup Pending state because the table space state is changed at the beginning of the load operation and cannot be rolled back.

Restore Pending

If you perform a successful load operation with the **COPY NO** option, restore the database, and then rollforward through that operation, the associated table spaces are placed in the Restore Pending state. To remove the table spaces from the Restore Pending state, you must perform a restore operation.

Note: DB2 LOAD does not set the table space state to **Load Pending** or **Delete Pending**.

Example of a table space state

If you load an input file (`staffdata.del`) into a table NEWSTAFF, as follows:

```
update db cfg for sample using logarchmeth1 logretain;
backup db sample;
connect to sample;
create table newstaff like staff;
load from staffdata.del of del insert into newstaff copy no;
connect reset;
```

and you open another session and issue the following commands,

```
connect to sample;
list tablespaces;
connect reset;
```

USERSPACE1 (the default table space for the sample database) is in the Load in Progress state and, after the first commit, the Backup Pending state as well. After the load operation finishes, the **LIST TABLESPACES** command reveals that USERSPACE1 is now in the Backup Pending state:

```
Tablespace ID                        = 2
Name                                 = USERSPACE1
Type                                 = Database managed space
Contents                             = All permanent data. Large table space.
State                                = 0x0020
  Detailed explanation:
  Backup pending
```

Table states during and after load operations

The load utility uses table states to preserve database consistency during a load operation. These states work by controlling access to data or eliciting user actions.

To determine the state of a table, issue the **LOAD QUERY** command, which also checks the status of a load operation. Tables can be in a number of states simultaneously. The states returned by **LOAD QUERY** are as follows:

Normal State
: The Normal state is the initial state of a table after it is created, indicating that no (abnormal) states currently affect the table.

Read Access Only
: If you specify the **ALLOW READ ACCESS** option, the table is in the Read Access Only state. The data in the table that existed before the invocation of the load command is available in read-only mode during the load operation. If you specify the **ALLOW READ ACCESS** option and the load operation fails, the data that existed in the table before the load operation continues to be available in read-only mode after the failure.

Load in Progress
: The Load in Progress table state indicates that there is a load in progress on the table. The load utility removes this transient state after the load is

successfully completed. However, if the load operation fails or is interrupted, the table state will change to Load Pending.

Redistribute in Progress

The Redistribute in Progress table state indicates that there is a redistribute in progress on the table. The redistribute utility removes this transient state after it has successfully completed processing the table. However, if the redistribute operation fails or is interrupted, the table state will change to Redistribute Pending.

Load Pending

The Load Pending table state indicates that a load operation failed or was interrupted. You can take one of the following steps to remove the Load Pending state:

- Address the cause of the failure. For example, if the load utility ran out of disk space, add containers to the table space. Then, restart the load operation.
- Terminate the load operation.
- Run a load **REPLACE** operation against the same table on which the load operation failed.
- Recover table spaces for the loading table by using the **RESTORE DATABASE** command with the most recent table space or database backup, then carry out further recovery actions.

Redistribute Pending

The Redistribute Pending table state indicates that a redistribute operation failed or was interrupted. You can perform a **REDISTRIBUTE CONTINUE** or **REDISTRIBUTE ABORT** operation to remove the Redistribute Pending state.

Not Load Restartable

In the Not Load Restartable state, a table is partially loaded and does not allow a load restart operation. There are two situations in which a table is placed in the Not Load Restartable state:

- If you perform a rollforward operation after a failed load operation that you could not successfully restart or terminate
- If you perform a restore operation from an online backup that you took while the table was in the Load in Progress or Load Pending state

The table is also in the Load Pending state. To remove the table from the Not Load Restartable state, issue the **LOAD TERMINATE** or the **LOAD REPLACE** command.

Set Integrity Pending

The Set Integrity Pending state indicates that the loaded table has constraints which have not yet been verified. The load utility places a table in this state when it begins a load operation on a table with constraints. Use the SET INTEGRITY statement to take the table out of Set Integrity Pending state.

Type-1 indexes

The Type-1 Indexes state indicates that the table currently uses type-1 indexes. Type-1 indexes are no longer supported since Version 9.7. You should convert them to type-2 indexes before upgrading to Version 10. Otherwise, the type-1 indexes are automatically rebuilt as type-2 indexes the first time a table is accessed.

For details on how to convert type-1 indexes before upgrading databases, see the “Converting type-1 indexes to type-2 indexes” topic.

Unavailable

Rolling forward through an unrecoverable load operation places a table in the Unavailable state. In this state, the table is unavailable; you must drop it or restore it from a backup.

Example of a table in multiple states

If you load an input file (`staffdata.del`) with a substantial amount of data into a table NEWSTAFF, as follows:

```
connect to sample;
create table newstaff like staff;
load from staffdata.del of del insert into newstaff allow read access;
connect reset;
```

and you open another session and issue the following commands,

```
connect to sample;
load query table newstaff;
connect reset;
```

the **LOAD QUERY** command reveals that the NEWSTAFF table is in the Read Access Only and Load in Progress table states:

```
Tablestate:
 Load in Progress
 Read Access Only
```

Load exception tables

A load exception table is a consolidated report of all of the rows that violated unique index rules, range constraints, and security policies during a load operation. You specify a load exception table by using the FOR EXCEPTION clause of the **LOAD** command.

Restriction: An exception table can not contain an identity column or any other type of generated column. If an identity column is present in the primary table, the corresponding column in the exception table should only contain the column's type, length, and nullability attributes. In addition, the exception table cannot be partitioned or have a unique index. Also, you cannot specify an exception table if:

- the target table uses LBAC security and has at least one XML column.
- the target table is range partitioned and has at least one XML column.

The exception table used with the load utility is identical to the exception tables used by the SET INTEGRITY statement. It is a user-created table that reflects the definition of the table being loaded and includes some additional columns.

You can assign a load exception table to the table space where the table being loaded resides or to another table space. In either case, assign the load exception table and the table being loaded to the same database partition group, and ensure that both tables use the same distribution key. Additionally, ensure that the exception table and table being loaded have the same partition map id (SYSIBM.SYSTABLES.PMAP_ID), which can potentially be different during the redistribute operation (add/drop database partition operation).

When to use an exception table

Use an exception table when loading data that has a unique index and could have duplicate records. If you do not specify an exception table and duplicate records are found, the load operation continues, and only a warning message is issued about the deleted duplicate records. The duplicate records are not logged.

After the load operation is completed, you can use information in the exception table to correct data that is in error. You can then insert the corrected data into the table.

Rows are appended to existing information in the exception table. Because there is no checking done to ensure that the table is empty, new information is simply added to the invalid rows from previous load operations. If you want only the invalid rows from the current load operation, you can remove the existing rows before invoking the utility. Alternatively, when you define a load operation, you can specify that the exception table record the time when a violation is discovered and the name of the constraint violated.

Because each deletion event is logged, the log could fill up during the delete phase of the load if there are a large number of records that violate a uniqueness condition.

Any rows rejected because of invalid data before the building of an index are not inserted into the exception table.

Monitoring a load operation using the LIST UTILITIES command

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

Procedure

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

```
list utilities show detail
```

Example

The following is an example of the output for monitoring the performance of a load operation using the **LIST UTILITIES** command:

```
ID                                = 10
Type                              = LOAD
Database Name                     = TEST
Member Number                     = 1
Description                       = OFFLINE LOAD DEL AUTOMATIC INDEXING REPLACE
COPY NO BEER   .TABLE1
Start Time                        = 08/16/2011 08:52:53.861841
State                             = Executing
Invocation Type                   = User
Progress Monitoring:
   Phase Number                   = 1
      Description                 = SETUP
      Total Work                  = 0 bytes
      Completed Work              = 0 bytes
      Start Time                  = 08/16/2011 08:52:53.861865

   Phase Number [Current]         = 2
      Description                 = LOAD
      Total Work                  = 49900 rows
      Completed Work              = 25313 rows
      Start Time                  = 08/16/2011 08:52:54.277687
```

```
Phase Number                  = 3
   Description                = BUILD
   Total Work                 = 2 indexes
   Completed Work             = 0 indexes
   Start Time                 = Not Started
```

Chapter 34. Ingest utility

The ingest utility (sometimes referred to as continuous data ingest, or CDI) is a high-speed client-side DB2 utility that streams data from files and pipes into DB2 target tables. Because the ingest utility can move large amounts of real-time data without locking the target table, you do not need to choose between the data currency and availability.

The ingest utility ingests pre-processed data directly or from files output by ETL tools or other means. It can run continually and thus it can process a continuous data stream through pipes. The data is ingested at speeds that are high enough to populate even large databases in partitioned database environments.

An **INGEST** command updates the target table with low latency in a single step. The ingest utility uses row locking, so it has minimal interference with other user activities on the same table.

With this utility, you can perform DML operations on a table using a SQL-like interface without locking the target table. These ingest operations support the following SQL statements: INSERT, UPDATE, MERGE, REPLACE, and DELETE. The ingest utility also supports the use of SQL expressions to build individual column values from more than one data field.

Other important features of the ingest utility include:

- **Commit by time or number of rows.** You can use the **commit_count** ingest configuration parameter to have commit frequency determined by the number of written rows or use the default **commit_period** ingest configuration parameter to have commit frequency determined by a specified time.
- **Support for copying rejected records to a file or table, or discarding them.** You can specify what the **INGEST** command does with rows rejected by the ingest utility (using the **DUMPFILE** parameter) or by DB2 (using the **EXCEPTION TABLE** parameter).
- **Support for restart and recovery.** By default, all **INGEST** commands are restartable from the last commit point. In addition, the ingest utility attempts to recover from certain errors if you have set the **retry_count** ingest configuration parameter.

The **INGEST** command supports the following input data formats:

- Delimited text
- Positional text and binary
- Columns in various orders and formats

In addition to regular tables and nicknames, the **INGEST** command supports the following table types:

- multidimensional clustering (MDC) and insert time clustering (ITC) tables
- range-partitioned tables
- range-clustered tables (RCT)
- materialized query tables (MQTs) that are defined as MAINTAINED BY USER, including summary tables
- temporal tables

- updatable views (except typed views)

A single **INGEST** command goes through three major phases:

1. Transport
The transporters read from the data source and put records on the formatter queues. For INSERT and MERGE operations, there is one transporter thread for each input source (for example, one thread for each input file). For UPDATE and DELETE operations, there is only one transporter thread.

2. Format
The formatters parse each record, convert the data into the format that DB2 database systems require, and put each formatted record on one of the flusher queues for that record's partition. The number of formatter threads is specified by the **num_formatters** configuration parameter. The default is (number of logical CPUs)/2.

3. Flush
The flushers issue the SQL statements to perform the operations on the DB2 tables. The number of flushers for each partition is specified by the **num_flushers_per_partition** configuration parameter. The default is max(1, ((number of logical CPUs)/2)/(number of partitions)).

Deciding where to run the ingest utility

The ingest utility is included as a part of the DB2 client install. You can run it from either the client or the server.

About this task

There are two choices for where to run the ingest utility:

On an existing server in the data warehouse environment
There are two choices for where to run ingest jobs within this type of setup:

- On the DB2 coordinator partition (the database partition server to which applications will connect and on which the coordinating agent is located)
- On an existing ETL (extract, transform, and load) server

On a new server
There are two choices for where to run ingest jobs within this type of setup:

- On a server that is only running the ingest utility
- On a server that is also hosting an additional DB2 coordinator partition that is dedicated to the ingest utility.

There are a number of factors that can influence where you decide to install the ingest utility:

- Performance: Having the ingest utility installed on its own server has a significant performance benefit, so this would be suitable for environments with large data sets.
- Cost: Having the ingest utility installed on an existing server means that no additional expenses are incurred as a result of using it.
- Ease of administration

Ingest-related tasks

This section provides a high-level overview of the main setup and operational tasks related to using the ingest utility.

Setting up ingest middleware

1. Decide where to run the ingest utility

   You can run ingest jobs on an existing machine or from a stand-alone machine. For more information, see “Deciding where to run the ingest utility” on page 594
2. Install the ingest utility (part of the DB2 Data Server Runtime Client and the DB2 Data Server Client).

   If you decide to install the ingest utility on a new, stand-alone machine, run the install for the DB2 client image. For more information, see “Installing IBM data server clients (Linux, UNIX)” in *Installing IBM Data Server Clients*

Developing a process to populate a table

1. (If required) Address code page issues

   Depending on whether the same code page is used by the input data, the DB2 client, and the DB2 server, there may be some user actions to take before running an **INGEST** command. For more information, see “Code page considerations for the ingest utility” on page 609.
2. Set up to handle the restart of failed **INGEST** commands

   To make an ingest operation restartable, you need to create a restart log table before issuing the **INGEST** command. For more information, see “Creating the restart table” on page 596.
3. Write an **INGEST** command

   Issue the **INGEST** command along with the mandatory parameters, like the input source and data type, and various optional parameters. For a detailed description of the command syntax and usage, as well as examples, see “Ingesting data” on page 597 and “INGEST” in the *Command Reference*.
4. Set up to process an ongoing stream of ingest jobs

   If you want to easily call a pre-written **INGEST** command, create a script for the command and call it when necessary. For more information, see “Scenario: Processing a stream of files with the ingest utility” on page 612

Performing operational tasks

- (If required) Addressing a failed **INGEST** command

  If an ingest job fails, you have the option of restarting or terminating the command. For more information, see “Restarting a failed ingest operation” on page 604 or “Terminating a failed ingest operation” on page 606.
- Monitoring an **INGEST** command

  For more information, see “Monitoring ingest operations” on page 613.

(Optional) Optimizing performance

- Review tunable configuration parameters for the **INGEST** command.
- Modify your **INGEST** command to meet high performance requirements.

  For more information, see “Performance considerations for ingest operations” on page 608.

Creating the restart table

By default, failed **INGEST** commands are restartable from the last commit point; however you first need to create a restart table, which stores the information needed to resume an **INGEST** command.

About this task

You have to create the restart table only once, and that table will be used by all **INGEST** commands in the database.

The ingest utility will use this table to store information needed to resume an incomplete INGEST command from the last commit point.

Note: The restart table does not contain copies of the input rows, only some counters to indicate which rows have been committed.

Restrictions

- It is recommended that you place the restart table in the same tablespace as the target tables that the ingest utility updates. If this is not possible, you must ensure that the tablespace containing the restart table is at the same level as the tablespace containing the target table. For example, if you restore or roll forward one of the table spaces, you must restore or roll forward the other to the same level. If the table spaces are at different levels and you run an **INGEST** command with the **RESTART CONTINUE** option, the ingest utility could fail or ingest incorrect data.
- If your disaster recovery strategy includes replicating the target tables of ingest operations, you must also replicate the restart table so it is kept in sync with the target tables.

Procedure

To create the restart table:

- If you are using a Version 10.1 server, call the SYSPROC.SYSINSTALLOBJECTS stored procedure:

  ```
  db2 "CALL SYSPROC.SYSINSTALLOBJECTS('INGEST', 'C', tablespace-name, NULL)"
  ```

- If you are using a Version 9.5, Version 9.7, or Version 9.8 server, issue the following SQL statements:

  ```
  CREATE TABLE SYSTOOLS.INGESTRESTART (
    JOBID            VARCHAR(256) NOT NULL,
    APPLICATIONID    VARCHAR(256) NOT NULL,
    FLUSHERID        INT          NOT NULL,
    FLUSHERDISTID    INT          NOT NULL,
    TRANSPORTERID    INT          NOT NULL,
    BUFFERID         BIGINT       NOT NULL,
    BYTEPOS          BIGINT       NOT NULL,
    ROWSPROCESSED    INT          NOT NULL,
      PRIMARY KEY (JOBID, FLUSHERID, TRANSPORTERID, FLUSHERDISTID))
    IN <tablespace-name>
    DISTRIBUTE BY (FLUSHERDISTID);

  GRANT SELECT, INSERT, UPDATE, DELETE
     ON TABLE SYSTOOLS.INGESTRESTART TO PUBLIC;
  ```

Results

The restart table, SYSTOOLS.INGESTRESTART, should now be created in the specified table space, and you can now run restartable **INGEST** commands.

Example

A DBA intends to run all **INGEST** commands as restartable, so the DBA needs to first create a restart table:

1. The DBA connects to the database:
   ```
   db2 CONNECT TO sample
   ```
2. The DBA calls the stored procedure:
   ```
   db2 "CALL SYSPROC.SYSINSTALLOBJECTS('INGEST', 'C', NULL, NULL)"
   ```

What to do next

Ensure that any user who will modify the restart table has the appropriate authorization:

- If the **INGEST** command specifies RESTART NEW, the user must have SELECT, INSERT, UPDATE, and DELETE privilege on the restart table.
- If the **INGEST** command specifies RESTART TERMINATE, the user must have SELECT and DELETE privilege on the restart table.

Ingesting data

You can use the ingest utility to continuously pump data into DB2 tables using SQL array inserts, updates, and deletes until sources are exhausted.

Before you begin

Before invoking the ingest utility, you must be connected to the database into which the data will be imported.

By default, failed **INGEST** commands are restartable from the last commit point; however you must first create a restart table, otherwise you receive an error message notifying you that the command you issued is not restartable. The ingest utility uses this table to store information needed to resume an incomplete **INGEST** command from the last commit point. For more information about this, see “Creating the restart table” on page 596.

About this task

For a list of the required privileges and authorities, see the **INGEST** command authorization.

Restrictions

For a comprehensive list of restrictions for the ingest utility, see “Ingest utility restrictions and limitations” on page 606.

Procedure

Issue the **INGEST** command specifying, at a minimum, a source, the format, and the target table as in the following example:

```
INGEST FROM FILE my_file.txt
    FORMAT DELIMITED
    INSERT INTO my_table;
```

It is recommended that you also specify a string with the **RESTART NEW** parameter on the **INGEST** command:

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   RESTART NEW 'CDIjob001'
   INSERT INTO my_table;
```

The string you specify can be up to 128 bytes. Because the string uniquely identifies the **INGEST** command, it must be unique across all **INGEST** commands in the current database that specified the **RESTART NEW** option and are not yet complete.

Example

Basic ingest examples

The following example inserts data from a delimited text file:

```
INGEST FROM FILE my_file.txt
    FORMAT DELIMITED
    INSERT INTO my_table;
```

The following example inserts data from a delimited text file with fields separated by a comma (the default). The fields in the file correspond to the table columns.

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 DATE 'mm/dd/yyyy',
      $field3 CHAR(32)
   )
   INSERT INTO my_table
      VALUES($field1, $field2, $field3);
```

Delimiter override example

The following example inserts data like the previous example, but the fields are separated by a vertical bar.

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED by '|'
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

Omitting the field definition and VALUES list example

In the following example, the table is defined as follows:

```
CREATE TABLE my_table (
    c1 VARCHAR(32),
    c2 INTEGER GENERATED BY DEFAULT AS IDENTITY,
    c3 INTEGER GENERATED ALWAYS AS (c2 + 1),
   );
```

The user issues the following **INGEST** command:

```
INGEST FROM FILE my_file.txt
    FORMAT DELIMITED
    INSERT INTO mytable;
```

- The default field definition list will be:

  ```
  (
      $C1 CHARACTER(32),
      $C2 INTEGER EXTERNAL,
      $C3 INTEGER EXTERNAL
  )
  ```

- The default VALUES list on the INSERT statement is:

  ```
  VALUES($C1, $C2, DEFAULT)
  ```

  Note that the third value is DEFAULT because the column that corresponds to field $C3 is defined as GENERATED ALWAYS. The fourth value is omitted because it has no field.

Extra fields used to compute column values example

The following example is the same as the delimiter override example, but only the first two fields correspond to the first two table columns (PROD_ID and DESCRIPTION), whereas the value for the third table column (TOTAL_PRICE) is computed from the remaining three fields

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED BY '|'
 (
  $prod_ID     CHAR(8),
  $description CHAR(32),
  $price       DECIMAL(5,2) EXTERNAL,
  $sales_tax   DECIMAL(4,2) EXTERNAL,
  $shipping    DECIMAL(3,2) EXTERNAL
 )
 INSERT INTO my_table(prod_ID, description, total_price)
  VALUES($prod_id, $description, $price + $sales_tax + $shipping);
```

Filler fields example

The following example inserts data from a delimited text file with fields separated by a comma (the default). The fields in the file correspond to the table columns except that there are extra fields between the fields for columns 2 and 3 and columns 3 and 4.

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1  INTEGER,
  $field2  CHAR(8),
  $filler1 CHAR,
  $field3  CHAR(32),
  $filler2 CHAR,
  $field4  DATE
 )
 INSERT INTO my_table VALUES($field1, $field2, $field3, $field4);
```

Format modifiers example

The following example inserts data from a delimited text file in code page 850. Date fields are in American format and char fields are enclosed in equal signs.

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 INPUT CODEPAGE 850
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
```

```
 $field3 CHAR(32) ENCLOSED BY '='
)
INSERT INTO my_table
 VALUES($field1, $field2, $field3);
```

Positional example

The following example inserts data from a file with fields in the specified positions. The fields in the file correspond to the table columns.

```
INGEST FROM FILE my_file.txt
    FORMAT POSITIONAL
 (
  $field1 POSITION(1:8)   INTEGER EXTERNAL,
  $field2 POSITION(10:19) DATE 'yyyy-mm-dd',
  $field3 POSITION(25:34) CHAR(10)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

DEFAULTIF examples

This example is similar to the previous example, except if the second field starts with a blank, the ingest utility inserts the default value:

```
INGEST FROM FILE my_file.txt
 FORMAT POSITIONAL
 (
  $field1 POSITION(1:8) INTEGER EXTERNAL,
  $field2 POSITION(10:19) DATE 'yyyy-mm-dd' DEFAULTIF = ' ',
    $field3 POSITION(25:34) CHAR(10)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

This example is the same as the previous example, except that the default indicator is in the column after the data columns:

```
INGEST FROM FILE my_file.txt
 FORMAT POSITIONAL
 (
  $field1 POSITION(1:8)   INTEGER EXTERNAL,
  $field2 POSITION(10:19) DATE 'yyyy-mm-dd' DEFAULTIF(35) = ' ',
    $field3 POSITION(25:34) CHAR(10)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

Multiple input sources example

This example inserts data from three delimited text files:

```
INGEST FROM FILE my_file.txt, my_file2.txt, my_file3.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

Pipe example

This example inserts data from a pipe:

```
INGEST FROM PIPE my_pipe
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

Options example

This example inserts data from a delimited text file with fields separated by a comma (the default). The fields in the file correspond to the table columns. The command specifies that write rows rejected by DB2 (for example, due to constraint violations) are to be written to table EXCP_TABLE, rows rejected due to other errors are to be discarded, and messages are to be written to file `messages.txt`.

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 EXCEPTION TABLE excp_table
    MESSAGES messages.txt
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

Restart example

This example issues an **INGEST** command (which is restartable, by default) with a specified ingest job id:

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 RESTART NEW 'ingestcommand001'
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

If the command terminates before completing, you can restart it with the following command:

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 RESTART CONTINUE 'ingestcommand001'
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

Restart terminate example

This example issues the same **INGEST** command as the previous "Restart example":

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 RESTART NEW 'ingestcommand001'
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

If the command terminates before completing and you do not plan to restart it, you can clean up the restart records with the following command.

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 RESTART TERMINATE 'ingestcommand001'
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

After issuing this command, you can no longer restart the **INGEST** command with the job id: "ingestcommand001", but you can reuse that string on the RESTART NEW parameter of a new **INGEST** command.

Reordering columns example

This example inserts data from a delimited text file with fields separated by a comma. The table has three columns and the fields in the input data are in the reverse order of the table columns.

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 INSERT INTO my_table
  VALUES($field3, $field2, $field1);
```

Basic UPDATE, MERGE, and DELETE examples

The following examples update the table rows whose primary key matches the corresponding fields in the input file.

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key1  INTEGER EXTERNAL,
  $key2  INTEGER EXTERNAL,
  $data1 CHAR(8),
  $data2 CHAR(32),
  $data3 DECIMAL(5,2) EXTERNAL
  )
 UPDATE my_table
  SET (data1, data2, data3) = ($data1, $data2, $data3)
  WHERE (key1 = $key1) AND (key2 = $key2);
```

or

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key1  INTEGER EXTERNAL,
  $key2  INTEGER EXTERNAL,
  $data1 CHAR(8),
  $data2 CHAR(32),
  $data3 DECIMAL(5,2) EXTERNAL
  )
 UPDATE my_table
  SET data1 = $data1, data2 = $data2, data3 = $data3
  WHERE (key1 = $key1) AND (key2 = $key2);
```

This example merges data from the input file into the target table. For input rows whose primary key fields match a table row, it updates that table row with the input row. For other input rows, it adds the row to the table.

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key1  INTEGER EXTERNAL,
  $key2  INTEGER EXTERNAL,
  $data1 CHAR(8),
  $data2 CHAR(32),
  $data3 DECIMAL(5,2) EXTERNAL
  )
 MERGE INTO my_table
  ON (key1 = $key1) AND (key2 = $key2)
  WHEN MATCHED THEN
   UPDATE SET (data1, data2, data3) = ($data1, $data2, $data3)
  WHEN NOT MATCHED THEN
   INSERT VALUES($key1, $key2, $data1, $data2, $data3);
```

This example deletes table rows whose primary key matches the corresponding primary key fields in the input file.

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key1  INTEGER EXTERNAL,
  $key2  INTEGER EXTERNAL
  )
 DELETE FROM my_table
  WHERE (key1 = $key1) AND (key2 = $key2);
```

Complex SQL examples

Consider the following example in which there is a table with columns KEY, DATA, and ACTION. The following command updates the DATA column of table rows where the primary key column (KEY) matches the corresponding field in the input file and the ACTION column is 'U':

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key_fld  INTEGER EXTERNAL,
  $data_fld INTEGER EXTERNAL
 )
 UPDATE my_table
  SET data = $data_fld
   WHERE (key = $key_fld) AND (action = 'U');
```

The following example is the same as the previous example except that if the keys match and the ACTION column is 'D', then it deletes the row from the table:

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key_fld  INTEGER EXTERNAL,
  $data_fld INTEGER EXTERNAL
 )
 MERGE INTO my_table
  ON (keyl = $key_fld)
  WHEN MATCHED AND (action = 'U') THEN
   UPDATE SET data = $data_fld
  WHEN MATCHED AND (action = 'D') THEN
   DELETE;
```

What to do next

If the **INGEST** command completes successfully, you can reuse the string specified with the **RESTART NEW** parameter.

If the **INGEST** command fails and you want to restart it, you must specify the **RESTART CONTINUE** option with the string you specified in the original command.

If you do not plan to restart the failed **INGEST** command and you want to clean up the entries in the restart table, rerun the **INGEST** command, specifying the **RESTART TERMINATE** option.

Restarting a failed ingest operation

If an **INGEST** command fails before completing and you want to restart it, reissue the **INGEST** command with the RESTART CONTINUE option. This second **INGEST** command starts from the last commit point and is also restartable.

Before you begin

The userid restarting the failed **INGEST** command must have SELECT, INSERT, UPDATE, and DELETE privilege on the restart log table.

About this task

The **INGEST** utility considers a command to be complete when it reaches the end of the file or pipe. Under any other conditions, the **INGEST** utility considers the command incomplete. These can include:

- The **INGEST** command gets an I/O error while reading the input file or pipe.
- The **INGEST** command gets a critical system error from the DB2 database system.
- The **INGEST** command gets a DB2 database system error that is likely to prevent any further SQL statements in the **INGEST** command from succeeding (for example, if the table no longer exists).
- The **INGEST** command is killed or terminates abnormally.

Restrictions

- If the target table and the restart table are in different table spaces, the two table spaces must be at the same level in terms of rollforward or restore operations.
- You cannot modify the contents of the restart table, other than restoring the entire table to keep it in sync with the target table.
- The **num_flushers_per_partition** configuration parameter must be the same as on the original command.
- If the input is from files or pipes, the number of input files or pipes must be the same as on the original command.
- The input file or pipes must provide the same records and in the same order as on the original command.
- The following **INGEST** command parameters must be the same as on the original command:
  - input type (file or pipe)
  - the SQL statement
  - the field definition list, including the number of fields and all field attributes
- The target table columns that the SQL command updates must have the same definition as they had at the time of the original command.

- In a partitioned database environment, you cannot have added or removed database partitions.
- In a partitioned database environment, you cannot have redistributed data across the partitions.
- If an **INGEST** command specifies the DUMPFILE (BADFILE) parameter, the dump file is guaranteed to be complete only if the **INGEST** command completes normally in a single run. If an **INGEST** command fails and the restarted command succeeds, the combination of dump files from the two commands might be missing some records or might contain duplicate records.

If the third, fourth, fifth, or ninth restriction is violated, the ingest utility issues an error and ends the **INGEST** command. In the case of the other restrictions, the ingest utility does not issue an error, but the restarted **INGEST** command might produce different output rows than the original would have if it had completed.

Procedure

To restart a failed **INGEST** operation, do the following:

1. Use the available information to diagnose and correct the problem that caused the failure
2. Reissue the **INGEST** command, specifying the **RESTART CONTINUE** option with the appropriate job-id.

Results

Once the restarted **INGEST** command completes, you can reuse the job-id on a later **INGEST** command.

Example

The following **INGEST** command failed:

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 DATE 'mm/dd/yyyy',
      $field3 CHAR(32)
   )
   RESTART NEW 'ingestjob001'
   INSERT INTO my_table
      VALUES($field1, $field2, $field3);
```

The DBA corrects the problem that cause the failure and restarts the **INGEST** command (which starts from the last commit point) with the following command:

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 DATE 'mm/dd/yyyy',
      $field3 CHAR(32)
   )
   RESTART CONTINUE 'ingestjob001'
   INSERT INTO my_table
      VALUES($field1, $field2, $field3);
```

Terminating a failed ingest operation

If an **INGEST** command fails before completing and you do not want to restart it, reissue the **INGEST** command with the **RESTART TERMINATE** option. This command option cleans up the log records for the failed **INGEST** command.

Before you begin

The user ID terminating the failed **INGEST** command must have SELECT and DELETE privilege on the restart log table.

Procedure

To terminate a failed **INGEST** operation, reissue the **INGEST** command. Specify the **RESTART TERMINATE** parameter with the appropriate string.

Results

After the restarted **INGEST** command completes, you can reuse the **RESTART NEW** string on a later **INGEST** command.

Example

The following **INGEST** command failed:

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 DATE 'mm/dd/yyyy',
      $field3 CHAR(32)
   )
   RESTART NEW 'ingestjob001'
   INSERT INTO my_table
      VALUES($field1, $field2, $field3);
```

The DBA does not want to restart the **INGEST** command, so they terminate it with the following command (which includes the **RESTART TERMINATE** parameter):

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 DATE 'mm/dd/yyyy',
      $field3 CHAR(32)
   )
   RESTART TERMINATE 'ingestjob001'
   INSERT INTO my_table
      VALUES($field1, $field2, $field3);
```

Ingest utility restrictions and limitations

There are a number of restrictions that you should be aware of when using the ingest utility.

Restartability

- If input data source type changed, the ingest utility might not be able to detect the change and will produce different output rows than the original failed command.

Table support

- The ingest utility supports operations against only DB2 for Linux, UNIX and Windows tables.
- The ingest utility does not support operations on:
  - created or declared global temporary tables
  - typed tables
  - typed views

Input types, formats, and column types

- The ingest utility does not support the following column types:
  - large object types (LOB, BLOB, CLOB, DBCLOB)
  - XML
  - structured types
  - columns with a user-defined data type based on any of the types listed previously
- In addition, the ingest utility has the following restrictions on generated columns:
  - The ingest utility cannot assign a value to a column defined as GENERATED ALWAYS. If the SQL statement on the **INGEST** command is INSERT or UPDATE and the target table has a GENERATED ALWAYS column, the insert or update operation fails (SQL0798N) and the **INGEST** command ends unless you do one of the following:
    - Omit the column from the list of columns to update.
    - On the INSERT or UPDATE statement, specify DEFAULT as the value assigned to the column.
  - The ingest utility cannot assign a combination of default values and specific values to a column defined as GENERATED BY DEFAULT AS IDENTITY. If the SQL statement on the **INGEST** command is INSERT or UPDATE and the target table has a GENERATED BY DEFAULT AS IDENTITY column, the insert or update operation fails (SQL0407N) and the **INGEST** command rejects the record unless you do one of the following:
    - Omit the column from the list of columns to update.
    - On the INSERT or UPDATE statement, specify DEFAULT as the value assigned to the column.
    - Specify an expression that never evaluates to NULL as the value assigned to the column. For example, if the expression is $field1, then $field1 can never have a NULL value in the input records.

Restrictions related to using other DB2 features with the ingest utility

- Except for the **CONNECT_MEMBER** parameter, the **SET CLIENT** command (for connection settings) does not affect how the ingest utility connects.
- The **LIST HISTORY** command does not display ingest operations.
- The **SET UTIL_IMPACT_PRIORITY** command does not affect the **INGEST** command
- The **util_impact_lim** database manager configuration parameter does not affect the **INGEST** command
- Except for CURRENT SCHEMA, CURRENT TEMPORAL SYSTEM_TIME, and CURRENT TEMPORAL BUSINESS_TIME, the ingest utility ignores the settings of most special registers that affect SQL statement execution.

General ingest utility restrictions

- If you ingest into a view that has multiple base tables, any base tables that are protected by a security policy must be protected by the same security policy. (You can still have some base tables unprotected but those that are protected must use the same security policy.)

Nickname support

- If the **INGEST** command specifies or defaults to the RESTART NEW or RESTART CONTINUE option, and the target table is a nickname or an updatable view that updates a nickname, ensure that the DB2_TWO_PHASE_COMMIT server option is set to 'Y' for the server definition that contains the nickname.
- You cannot use the **SET SERVER OPTION** to enable two-phase commit before issuing the **INGEST** command because that command affects only the CLP connection, whereas the **INGEST** command establishes its own connection. You must set the server option in the server definition in the catalog.
- You cannot use the DB2_TWO_PHASE_COMMIT server option with the database partitioning feature, which means that the combination of partitioned database environment mode, a restartable ingest command, and ingesting into a nickname is not supported.
- The performance benefit of the utility is reduced when used on nicknames.

Performance considerations for ingest operations

Use the following set of guidelines to help performance tune your ingest jobs.

Field type and column type

Define fields to be the same type as their corresponding column types. When the types are different, the ingest utility or DB2 must convert the input data to the column type.

Materialized query tables (MQTs)

If you ingest data into a table that is a base table of an MQT defined as REFRESH IMMEDIATE, performance can degrade significantly due to the time required to update the MQT.

Row size

For tables with a small row size, increase the setting of the **commit_count** ingest configuration parameter; for tables with a large row size, reduce the setting of the **commit_count** ingest configuration parameter.

Other workloads

If you are executing the ingest utility with another workload, increase the setting of the **locklist** database configuration parameter and reduce the setting of the **commit_count** ingest configuration parameter

Code page considerations for the ingest utility

When the ingest utility processes input data, there are three code pages involved: the application (client) code page, the input data code page, and the database code page.

Code page	How specified	Default
Application (client) code page, which is used in the CLP command file	Determined from the current locale	Determined from the current locale
Input data code page	INPUT CODEPAGE on the **INGEST** command	Application code page
Database code page	Specified on the **CREATE DATABASE** command	1208 (UTF-8 encoding of Unicode)

If the input data code page differs from the application code page, the ingest utility temporarily overrides the application code page with the input data code page so that DB2 converts the data directly from the input data code page to the database code page. Under some conditions, the ingest utility cannot override the application code page. In this case, the ingest utility converts character data that is not defined as FOR BIT DATA to the application code page before passing it to DB2. In all cases, if the column is not defined as FOR BIT DATA, DB2 converts the data to the database code page.

CLP command file code page

Except for hex constants, the ingest utility assumes that the text of the **INGEST** command is in the application code page. Whenever the ingest utility needs to compare strings specified on the **INGEST** command (for example, when comparing the DEFAULTIF character to a character in the input data), the ingest utility performs any necessary code page conversion to ensure the compared strings are in the same code page. Neither the ingest utility nor DB2 do any conversion of hex constants.

Input data code page

If both a field and the table column that it is assigned to are defined as FOR BIT DATA, then neither the ingest utility nor DB2 does any code page conversion. For example, suppose that the **INGEST** command assigns field $c1 to column C1 and both are defined as CHAR FOR BIT DATA. If the input field contains X'E9', then DB2 sets column C1 to X'E9', regardless of the input data code page or database code page.

It is strongly recommended that if a column definition omits FOR BIT DATA, then its corresponding field definition also omit FOR BIT DATA. Likewise, if a column definition specifies FOR BIT DATA, its corresponding field should also specify FOR BIT DATA. Otherwise, the value assigned to the column is unpredictable because it depends on whether the ingest utility can override the application code page.

The following example illustrates this situation:

- The input data code page is 819.
- The application code page is 850.
- The database code page is 1208 (UTF-8).
- The input data is "é" ("e" with an acute accent), which is X'E9' in code page 819, X'82' in code page 850, and X'C3A9' in UTF-8.

The following table shows what data ends up on the server depending on whether the field and/or column are defined as FOR BIT DATA and whether the ingest utility can override the application code page:

Table 108. Possible outcomes if the field and column definitions are defined as FOR BIT DATA

Field definition	Column definition	Input data (code page 819)	Data after the ingest utility converts it to application code page 850	Data on the server if the ingest utility can override the application code page	Data on the server if the ingest utility cannot override the application code page
CHAR	CHAR	X'E9'	X'82'	X'C3A9'	X'C3A9'
CHAR FOR BIT DATA	CHAR FOR BIT DATA	X'E9'	X'E9'	X'E9'	X'E9'
CHAR FOR BIT DATA	CHAR	X'E9'	X'E9'	**X'C3A9'**	**X'C39A' ("Ú")**
CHAR	CHAR FOR BIT DATA	X'E9'	X'82'	**X'E9'**	**X'82'**

The data in the fourth column is what the ingest utility sends to DB2 when it can override the application code page. The data in the fourth column is what the ingest utility sends when it cannot override the application code page. Note that when the FOR BIT DATA attribute of the field and column definitions are different, the results can vary as shown in the preceding table.

Code page errors

In cases where the input code page, application code page, or database code page differ, either the ingest utility or DB2 or both will perform code page conversion. If DB2 does not support the code page conversion in any of the following cases, the ingest utility issues an error and the command ends.

Conversion is required when...	In this case, conversion from...	To...	Is done by...
The `INGEST` command contains strings or SQL identifiers that need to be converted to the input data code page.	Application code page	Input data code page	Ingest utility
The utility can override the application code page to be the input data code page.	Input code page	Database code page	DB2
The utility cannot override the application code page.	Input code page	Application code page	Ingest utility
The utility cannot override the application code page.	Application code page	Database code page	DB2

Ingest operations in a partitioned database environment

You can use the ingest utility to move data into a partitioned database environment.

INGEST commands running on a partitioned database use one or more flushers for each partition, as specified by the **num_flushers_per_partition** configuration parameter. The default is as follows:

```
max(1, ((number of logical CPUs)/2)/(number of partitions) )
```

You can also set this parameter to 0, meaning one flusher for all partitions.

Each flusher connects directly to the partition to which it will send data. In order for the connection to succeed, all the DB2 server partitions must use the same port number to receive client connections.

If the target table is a type that has a distribution key, the ingest utility determines the partition that each record belongs to as follows:

1. Determine whether every distribution key has exactly one corresponding field or constant value. This will be true if:
   - For an INSERT statement, the column list contains every distribution key and for each distribution key, the corresponding item in the VALUES list is a field name or a constant.
   - For an UPDATE or DELETE statement, the WHERE predicate is of the form

     ```
     (dist-key-col1 = value1) AND (dist-key-col2 = value2) AND ...
     (dist-key-coln = valuen) [AND any-other-conditions]
     ```

     where *dist-keycol1* to *dist-key-coln* are all the distribution keys and each *value* is a field name or a constant.
   - For a MERGE statement, the search condition is of the form shown previously for UPDATE and DELETE.
2. If every distribution key has exactly one corresponding field or constant value, the ingest utility uses the distribution key to determine the partition number and then routes the record to one of that partition's flushers.

   Note: In the following cases, the ingest utility does not determine the record's partition. If there is more than 1 flusher, the ingest utility routes the record to a flusher chosen at random:
   - The target table is a type that has no distribution key.
   - The column list (INSERT) or predicate (UPDATE, MERGE, DELETE) does not specify all distribution keys. In the following example, key columns 2-8 are missing:

     ```
     UPDATE my_table SET data = $data
         WHERE (key1 = $key1) AND (key9 = $key9);
     ```
   - A distribution key corresponds to more than one field or value, as in the following example:

     ```
     UPDATE my_table SET data = $data
         WHERE key1 = $key11 OR key1 = $key12;
     ```
   - A distribution key corresponds to an expression, as in the following example

     ```
     INGEST FROM FILE ...
         INSERT INTO my_table(dist_key, col1, col2)
             VALUES($field1 + $field2, $col1, $col2);
     ```
   - A distribution key column has type DB2SECURITYLABEL.

- A field that corresponds to a distribution key has a numeric type, but the distribution key column type is a different numeric type or has a different precision or scale.

Sample ingest utility scripts

You can use the ingest utility sample script to automate writing a new INGEST command each time there are new files to process.

The sample script `ingest_files.sh` is a shell script that automatically checks for new files and generates an INGEST command to process the files. The script performs the following tasks, in order:

1. Check the directory to see if there are new files to process. If there are no files, the script exits.

   Note: The script assumes that the specified directory only contains files for the table that you want to populate.
2. Obtain the names of the new files and then generate a separate INGEST command for each file
3. Run the INGEST command and handle the return code
4. Move the processed files to a success directory or a failed directory.

The script is provided in the `samples/admin_scripts` directory under your installation directory.

Modifying the script for your environment

You can use the `ingest_files.sh` script as a basis for your own script. The important modifications that you have to make to it are:

- Replace the sample values (namely, the database name, table name) with you own values
- Replace the sample INGEST command with your own command
- Create the directories specified in the script

The script processes files that contain data to populate a single table. To populate multiple tables, you can either replicate the mechanism for each table that you want to populate or generalize the mechanism to handle multiple tables.

Sample scenario

A sample scenario has been included in the documentation to show you how you can adapt the sample script to your data warehouse to automate the generation of new INGEST commands.

Scenario: Processing a stream of files with the ingest utility

The following scenario shows how you can configure your data warehouse to automatically ingest an ongoing stream of data files.

The problem: In some data warehouses, files arrive in an ongoing stream throughout the day and need to be processed as they arrive. This means that each time a new file arrives, another **INGEST** command needs to be run specifying the new file to process.

The solution: You can write a script that automatically checks for new files, generates a new **INGEST** command, and runs that command. The `ingest_files.sh` is a sample of such a script. You also need to create a crontab entry in order to specify how frequently the shell script is supposed to run.

Before the user implements this mechanism (that is, the script and the chrontab entry) for processing the stream of files, the user needs to have met the following prerequisites and dependencies:

- The target table has been created in the target database
- The ingest utility is ready to use (that is, it is installed and set up on a client machine)
- An INGEST command has been specified and verified by running it manually with a test file
- The objects, such as the exception table, referenced in the **INGEST** command have been created
- A crontab file has been created on the system on which the ingest utility is running
- The user has a process for creating the input files and moving them into the source directory that the script uses

1. The user creates a new script, using `ingest_files.sh` as a template by doing the following:
   a. Replace the following sample input values to reflect the user's values:
      - INPUT_FILES_DIRECTORY
      - DATABASE_NAME
      - SCHEMA_NAME
      - TABLE_NAME
      - SCRIPT_PATH

   b. Replace the sample **INGEST** command

   c. Save the script as `populate_table1_script`
2. The user adds an entry to the crontab file to specify how frequently the script is to run. Because the user wants the script to run once a minute, 24 hours a day, every day of the year, the user adds the following line:
   ```
   1 * * * * $HOME/bin/populate_table1_script
   ```
3. The user tests the script by creating new input files and adding them to the source directory.

Monitoring ingest operations

You can use the **INGEST LIST** or **INGEST GET STATS** commands to monitor the progress of **INGEST** commands.

Before you begin

To issue the **INGEST LIST** and **INGEST GET STATS** commands, you need a separate CLP session but they must be run on the same machine that the **INGEST** command is running on.

Procedure

There are a number of ways to monitor an ingest operation:

- To get basic information about all currently running **INGEST** commands, use the **INGEST LIST** command.
- To get more detailed information about a specific **INGEST** command or all currently running **INGEST** commands, use the **INGEST GET STATS** command.
- You can also query the following monitor elements by using an interface such as the MON_GET_CONNECTION table function:
  - **client_acctng**
  - **client_applname**
  - **appl_name**
  - **client_userid**
  - **client_wrkstnname**

Example

The following shows an example of what output to expect from an **INGEST LIST** command:

```
INGEST LIST

Ingest job ID       = DB21000:20101116.123456.234567:34567:45678
Ingest temp job ID  = 1
Database Name       = MYDB
Input type          = FILE
Target table        = MY_SCHEMA.MY_TABLE
Start Time          = 04/10/2010 11:54:45.773215
Running Time        = 01:02:03
Number of records processed = 30,000
```

The following shows an example of what output to expect from an **INGEST GET STATS** command:

```
INGEST GET STATS FOR 4

Ingest job ID = DB21000:20101116.123456.234567:34567:4567
Database      = MYDB
Target table  = MY_SCHEMA.MY_TABLE1

Records/sec      Flushes/sec       Records/sec     Flushes/sec
since start      since start       since last      since last       Total records
---------------- ----------------- --------------- ---------------- -------------
54321            65432             76543           87654            98765
```

The following shows an example of using the MON_GET_CONNECTION table function to get the number of rows modified and the number of commits:

```
SELECT client_acctng AS "Job ID",
      SUM(rows_modified) AS "Total rows modified",
      SUM(total_app_commits) AS "Total commits"
   FROM TABLE(MON_GET_CONNECTION(NULL, NULL))
   WHERE application_name = 'DB2_INGEST'
   GROUP BY client_acctng
   ORDER BY 1

Job ID                                          Total rows modified Total commits
----------------------------------------------- ------------------- -------------
DB21000:20101116.123456.234567:34567:45678                       92            52
DB21000:20101116.987654.234567:34567:45678                      172           132

  2 record(s) selected.
```

Chapter 35. Import utility

The import utility populates a table, typed table, or view with data using an SQL INSERT statement. If the table or view receiving the imported data already contains data, the input data can either replace or be appended to the existing data.

Like export, import is a relatively simple data movement utility. It can be activated by issuing CLP commands, by calling the ADMIN_CMD stored procedure, or by calling its API, db2Import, through a user application.

There are a number of data formats that import supports, as well as features that can be used with import:

- Import supports IXF, ASC, and DEL data formats.
- Import can be used with file type modifiers to customize the import operation.
- Import can be used to move hierarchical data and typed tables.
- Import logs all activity, updates indexes, verifies constraints, and fires triggers.
- Import allows you to specify the names of the columns within the table or view into which the data is to be inserted.
- Import can be used with DB2 Connect.

Import modes

Import has five modes which determine the method in which the data is imported. The first three, `INSERT`, `INSERT_UPDATE`, and `REPLACE` are used when the target tables already exist. All three support IXF, ASC, and DEL data formats. However, only `INSERT` and `INSERT_UPDATE` can be used with nicknames.

Table 109. Overview of INSERT, INSERT_UPDATE, and REPLACE import modes

Mode	Best practice usage
INSERT	Inserts input data into target table without changing existing data
INSERT_UPDATE	Updates rows with matching primary key values with values of input rows Where there's no matching row, inserts imported row into the table
REPLACE	Deletes all existing data and inserts imported data, while keeping table and index definitions

The other two modes, `REPLACE_CREATE` and `CREATE`, are used when the target tables do not exist. They can only be used with input files in the PC/IXF format, which contains a structured description of the table that is to be created. Imports cannot be performed in these modes if the object table has any dependents other than itself.

Note: Import's `CREATE` and `REPLACE_CREATE` modes are being deprecated. Use the **`db2look`** utility instead.

Table 110. Overview of REPLACE_CREATE and CREATE import modes

Mode	Best practice usage
REPLACE_CREATE	Deletes all existing data and inserts imported data, while keeping table and index definitions Creates target table and index if they don't exist
CREATE	Creates target table and index Can specify the name of the table space where the new table is created

In IBM Data Studio Version 3.1 or later, you can use the task assistant for importing data. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

How import works

The number of steps and the amount of time required for an import depend on the amount of data being moved and the options that you specify. An import operation follows these steps:

1. Locking tables
   Import acquires either an exclusive (X) lock or a nonexclusive (IX) lock on existing target tables, depending on whether you allow concurrent access to the table.
2. Locating and retrieving data
   Import uses the FROM clause to locate the input data. If your command indicates that XML or LOB data is present, import will locate this data.
3. Inserting data
   Import either replaces existing data or adds new rows of data to the table.
4. Checking constraints and firing triggers
   As the data is written, import ensures that each inserted row complies with the constraints defined on the target table. Information about rejected rows is written to the messages file. Import also fires existing triggers.
5. Committing the operation
   Import saves the changes made and releases the locks on the target table. You can also specify that periodic take place during the import.

The following items are mandatory for a basic import operation:

- The path and the name of the input file
- The name or alias of the target table or view
- The format of the data in the input file
- The method by which the data is to be imported
- The traverse order, when importing hierarchical data
- The subtable list, when importing typed tables

Additional options

There are a number of options that allow you to customize an import operation. You can specify file type modifiers in the MODIFIED BY clause to change the format of the data, tell the import utility what to do with the data, and to improve performance.

The import utility, by default, does not perform commits until the end of a successful import, except in the case of some ALLOW WRITE ACCESS imports. This improves the speed of an import, but for the sake of concurrency, restartability, and active log space considerations, it might be preferable to specify that commits take place during the import. One way of doing so is to set the **COMMITCOUNT** parameter to "automatic," which instructs import to internally determine when it should perform a commit. Alternatively, you can set **COMMITCOUNT** to a specific number, which instructs import to perform a commit once that specified number of records has been imported.

There are a few ways to improve import's performance. As the import utility is an embedded SQL application and does SQL fetches internally, optimizations that apply to SQL operations apply to import as well. You can use the compound file type modifier to perform a specified number of rows to insert at a time, rather than the default row-by-row insertion. If you anticipate that a large number of warnings will be generated (and, therefore, slow down the operation) during the import, you can also specify the norowwarnings file type modifier to suppress warnings about rejected rows.

Messages file

During an import, standard ASCII text message files are written to contain the error, warning, and informational messages associated with that operation. If the utility is invoked through the application programming interface (API) db2Import, you must specify the name of these files in advance with the **MESSAGES** parameter, otherwise it is optional. The messages file is a convenient way of monitoring the progress of an import, as you can access is while the import is in progress. In the event of a failed import operation, message files can be used to determine a restarting point by indicating the last row that was successfully imported.

Note: If the volume of output messages generated by an import operation against a remote database exceeds 60 KB, the utility will keep the first 30 KB and the last 30 KB.

Privileges and authorities required to use import

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects.

Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

With DATAACCESS authority, you can perform any type of import operation. The following table lists the other authorities on each participating table, view or nickname that enable you to perform the corresponding type of import.

Table 111. Authorities required to perform import operations

Mode	Required authority
INSERT	CONTROL or INSERT and SELECT
INSERT_UPDATE	CONTROL or INSERT, SELECT, UPDATE, and DELETE

Table 111. Authorities required to perform import operations (continued)

Mode	Required authority
REPLACE	CONTROL or INSERT, SELECT, and DELETE
REPLACE_CREATE	When the target table exists: CONTROL or INSERT, SELECT, and DELETE When the target table doesn't exist: CREATETAB (on the database), USE (on the table space), and when the schema does not exist: IMPLICIT_SCHEMA (on the database), or when the schema exists: CREATEIN (on the schema)
CREATE	CREATETAB (on the database), USE (on the table space), and when the schema does not exist: IMPLICIT_SCHEMA (on the database), or when the schema exists: CREATEIN (on the schema)

Note: The **CREATE** and **REPLACE_CREATE** options of the **IMPORT** command are deprecated and might be removed in a future release.
As well, to use the **REPLACE** or **REPLACE_CREATE** option on a table, the session authorization ID must have the authority to drop the table.

If you want to import to a hierarchy, the required authority also depends on the mode. For existing hierarchies, CONTROL privilege on every subtable in the hierarchy is sufficient for a **REPLACE** operation. For hierarchies that don't exist, CONTROL privilege on every subtable in the hierarchy, along with CREATETAB and USE, is sufficient for a **REPLACE_CREATE** operation.

In addition, there a few considerations for importing into tables with label-based access control (LBAC) security labels defined on them. To import data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. To import data into a table that has protected rows, the session authorization ID must have been granted a security label for write access that is part of the security policy protecting the table.

Importing data

The import utility inserts data from an external file with a supported file format into a table, hierarchy, view, or nickname.

The load utility is a faster alternative, but the load utility does not support loading data at the hierarchy level.

Before you begin

Before invoking the import utility, you must be connected to (or be able to implicitly connect to) the database into which you want to import the data. If implicit connect is enabled, a connection to the default database is established.

Utility access to DB2 for Linux, UNIX, or Windows database servers from DB2 for Linux, UNIX, or Windows clients must be a direct connection through the engine. Utility access cannot be through a DB2 Connect gateway or loop back environment.

Since the utility issues a COMMIT or a ROLLBACK statement, complete all transactions and release all locks by issuing a COMMIT statement or a ROLLBACK operation before invoking import.

Note: The **CREATE** and **REPLACE_CREATE** parameters of the **IMPORT** command are deprecated and might be removed in a future release.

Restrictions

The following restrictions apply to the import utility:

- If the existing table is a parent table containing a primary key that is referenced by a foreign key in a dependent table, its data cannot be replaced, only appended to.
- You cannot perform an import replace operation into an underlying table of a materialized query table defined in refresh immediate mode.
- You cannot import data into a system table, a summary table, or a table with a structured type column.
- You cannot import data into declared temporary tables.
- Views cannot be created through the import utility.
- Referential constraints and foreign key definitions are not preserved when creating tables from PC/IXF files. (Primary key definitions *are* preserved if the data was previously exported by using `SELECT *`.)
- Because the import utility generates its own SQL statements, the maximum statement size of 2 MB might, in some cases, be exceeded.
- You cannot re-create a partitioned table or a multidimensional clustered table (MDC) by using the **CREATE** or **REPLACE_CREATE** import parameters.
- You cannot re-create tables containing XML columns.
- You cannot import encrypted data.
- The import replace operation does not honor the Not Logged Initially clause. The **REPLACE** parameter for the **IMPORT** command does not honor the NOT LOGGED INITIALLY (NLI) clause for the CREATE TABLE statement clause or the ACTIVATE NOT LOGGED INITIALLY clause for the ALTER TABLE statement. If an import with the **REPLACE** action is performed within the same transaction as a CREATE TABLE or ALTER TABLE statement where the NLI clause is invoked, the import does not honor the NLI clause. In this scenario, all inserts are logged.

  Workaround 1: Delete the contents of the table by using the DELETE statement, then invoke the import with INSERT statement.

  Workaround 2: Drop the table and re-create it, then invoke the import with INSERT statement.

The following limitation applies to the import utility: If the volume of output messages generated by an import operation against a remote database exceeds 60 KB, the utility keeps the first 30 KB and the last 30 KB.

Procedure

To invoke the import utility:

- Issue an **IMPORT** command in the command line processor (CLP).
- Call the db2Import application programming interface (API) from a client application.
- Open the task assistant in IBM Data Studio for the **IMPORT** command.

Example

A simple import operation requires you to specify only an input file, a file format, an import mode, and a target table (or the name of the table that is to be created).

For example, to import data from the CLP, enter the **IMPORT** command:

```
db2 import from filename of fileformat import_mode into table
```

where *filename* is the name of the input file that contains the data you want to import, *fileformat* is the file format, *import_mode* is the mode, and *table* is the name of the table that you want to insert the data into.

However, you might also want to specify a messages file to which warning and error messages are written. To do that, add the **MESSAGES** parameter and a message file name. For example:

```
db2 import from filename of fileformat messages messagefile import_mode into table
```

Import sessions - CLP examples

Example 1

The following example shows how to import information frommyfile.ixf to the STAFF table:

```
   db2 import from myfile.ixf of ixf messages msg.txt insert into staff

SQL3150N  The H record in the PC/IXF file has product "DB2    01.00", date
"19970220", and time "140848".

SQL3153N  The T record in the PC/IXF file has name "myfile",
qualifier "        ", and source "            ".

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

SQL3110N  The utility has completed processing.  "58" rows were read from the
input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "58".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "58" rows were processed from the input file.  "58" rows were
successfully inserted into the table.  "0" rows were rejected.
```

Example 2

The following example shows how to import into a table that has identity columns:

TABLE1 has 4 columns:

- C1 VARCHAR(30)
- C2 INT GENERATED BY DEFAULT AS IDENTITY
- C3 DECIMAL(7,2)
- C4 CHAR(1)

TABLE2 is the same as TABLE1, except that C2 is a GENERATED ALWAYS identity column.

Data records in DATAFILE1 (DEL format):

```
"Liszt"
"Hummel",,187.43, H
"Grieg",100, 66.34, G
"Satie",101, 818.23, I
```

Data records in DATAFILE2 (DEL format):

```
"Liszt", 74.49, A
"Hummel", 0.01, H
"Grieg", 66.34, G
"Satie", 818.23, I
```

The following command generates identity values for rows 1 and 2, since no identity values are supplied in DATAFILE1 for those rows. Rows 3 and 4, however, are assigned the user-supplied identity values of 100 and 101, respectively.

```
db2 import from datafile1.del of del replace into table1
```

To import DATAFILE1 into TABLE1 so that identity values are generated for all rows, issue one of the following commands:

```
db2 import from datafile1.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
db2 import from datafile1.del of del modified by identityignore
   replace into table1
```

To import DATAFILE2 into TABLE1 so that identity values are generated for each row, issue one of the following commands:

```
db2 import from datafile2.del of del replace into table1 (c1, c3, c4)
db2 import from datafile2.del of del modified by identitymissing
   replace into table1
```

If DATAFILE1 is imported into TABLE2 without using any of the identity-related file type modifiers, rows 1 and 2 will be inserted, but rows 3 and 4 will be rejected, because they supply their own non-NULL values, and the identity column is GENERATED ALWAYS.

Example 3

The following example shows how to import into a table that has null indicators:

TABLE1 has 5 columns:

- COL1 VARCHAR 20 NOT NULL WITH DEFAULT
- COL2 SMALLINT
- COL3 CHAR 4
- COL4 CHAR 2 NOT NULL WITH DEFAULT
- COL5 CHAR 2 NOT NULL

ASCFILE1 has 6 elements:

- ELE1 positions 01 to 20
- ELE2 positions 21 to 22
- ELE5 positions 23 to 23
- ELE3 positions 24 to 27
- ELE4 positions 28 to 31
- ELE6 positions 32 to 32
- ELE6 positions 33 to 40

Data Records:

```
1...5....10...15...20...25...30...35...40
Test data 1         XXN 123abcdN
Test data 2 and 3   QQY    wxyzN
Test data 4,5 and 6 WWN6789    Y
```

The following command imports records from ASCFILE1 into TABLE1:

```
db2 import from ascfile1 of asc
method L (1 20, 21 22, 24 27, 28 31)
null indicators (0, 0, 23, 32)
insert into table1 (col1, col5, col2, col3)
```

Note:

1. Because COL4 is not provided in the input file, it will be inserted into TABLE1 with its default value (it is defined NOT NULL WITH DEFAULT).
2. Positions 23 and 32 are used to indicate whether COL2 and COL3 of TABLE1 will be loaded NULL for a given row. If there is a `Y` in the column's null indicator position for a given record, the column will be NULL. If there is an `N`, the data values in the column's data positions of the input record (as defined in L(........)) are used as the source of column data for the row. In this example, neither column in row 1 is NULL; COL2 in row 2 is NULL; and COL3 in row 3 is NULL.
3. In this example, the NULL INDICATORS for COL1 and COL5 are specified as 0 (zero), indicating that the data is not nullable.
4. The NULL INDICATOR for a given column can be anywhere in the input record, but the position must be specified, and the `Y` or `N` values must be supplied.

Typed table import considerations

The import utility can be used to move data both from and into typed tables while preserving the data's preexisting hierarchy. If desired, import can also be used to create the table hierarchy and the type hierarchy.

The movement of data from one hierarchical structure of typed tables to another is done through a specific traverse order and the creation of an intermediate flat file during an export operation. In turn, the import utility controls the size and the placement of the hierarchy being moved, using the **`CREATE`**, **`INTO`** *table-name*, **`UNDER`**, and **`AS ROOT TABLE`** parameters. As well, import determines what is placed in the target database. For example, it can specify an attributes list at the end of each subtable name to restrict the attributes that are moved to the target database. If no attributes list is used, all of the columns in each subtable are moved.

Table re-creation

The type of import you are able to perform depends on the file format of the input file. When working with ASC or DEL data, the target table or hierarchy must exist before the data can be imported. However, data from a PC/IXF file can be imported even if the table or hierarchy does not already exist if you specify an import **`CREATE`** operation. It must be noted that if the **`CREATE`** option is specified, import cannot alter subtable definitions.

Traverse order

The traverse order contained in the input file enables the hierarchies in the data to be maintained. Therefore, the same traverse order must be used when invoking the export utility and the import utility.

For the PC/IXF file format, one need only specify the target subtable name, and use the default traverse order stored in the file.

When using options other than **CREATE** with typed tables, the traverse order list enables one to specify the traverse order. This user-specified traverse order must match the one used during the export operation. The import utility guarantees the accurate movement of data to the target database given the following:

- An identical definition of subtables in both the source and the target databases
- An identical hierarchical relationship among the subtables in both the source and target databases
- An identical traverse order

Although you determine the starting point and the path down the hierarchy when defining the traverse order, each branch must be traversed to the end before the next branch in the hierarchy can be started. The import utility looks for violations of this condition within the specified traverse order.

Examples

Examples in this section are based on the following hierarchical structure with four valid traverse orders:

- Person, Employee, Manager, Architect, Student
- Person, Student, Employee, Manager, Architect
- Person, Employee, Architect, Manager, Student
- Person, Student, Employee, Architect, Manager

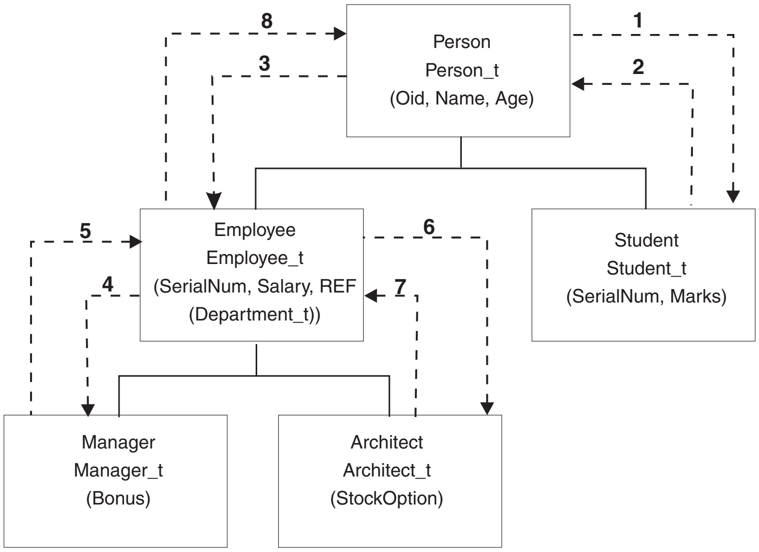


Figure 50. An example of a hierarchy

Example 1
To re-create an entire hierarchy (contained in the data file `entire_hierarchy.ixf` created by a prior export operation) using import, you would enter the following commands:

```
DB2 CONNECT TO Target_db
DB2 IMPORT FROM entire_hierarchy.ixf OF IXF CREATE INTO
   HIERARCHY STARTING Person AS ROOT TABLE
```

Each type in the hierarchy is created if it does not exist. If these types already exist, they must have the same definition in the target database as in the source database. An SQL error (SQL20013N) is returned if they are not the same. Since a new hierarchy is being created, none of the subtables defined in the data file being moved to the target database (Target_db) can exist. Each of the tables in the source database hierarchy is created. Data from the source database is imported into the correct subtables of the target database.

Example 2
To re-create the entire hierarchy of the source database and import it to the target database, while only keeping selected data, you would enter the following commands:

```
DB2 CONNECT TO Target_db
DB2 IMPORT FROM entire_hierarchy.del OF DEL INSERT INTO (Person,
   Employee(Salary), Architect) IN HIERARCHY (Person, Employee,
   Manager, Architect, Student)
```

The target tables PERSON, EMPLOYEE, and ARCHITECT must all exist. Data is imported into the PERSON, EMPLOYEE, and ARCHITECT subtables. That is, the following will be imported:

- All columns in PERSON into PERSON
- All columns in PERSON plus SALARY in EMPLOYEE into EMPLOYEE
- All columns in PERSON plus SALARY in EMPLOYEE, plus all columns in ARCHITECT into ARCHITECT

Columns SerialNum and REF(Employee_t) are not imported into EMPLOYEE or its subtables (that is, ARCHITECT, which is the only subtable having data imported into it).

Note: Because ARCHITECT is a subtable of EMPLOYEE, and the only import column specified for EMPLOYEE is SALARY, SALARY is also the only Employee-specific column imported into ARCHITECT. That is, neither SerialNum nor REF(Employee_t) columns are imported into either EMPLOYEE or ARCHITECT rows.
Data for the MANAGER and the STUDENT tables is not imported.

Example 3
This example shows how to export from a regular table, and import as a single subtable in a hierarchy. The **EXPORT** command operates on regular (non-typed) tables, so there is no Type_id column in the data file. The file type modifier `no_type_id` is used to indicate this, so that the import utility does not expect the first column to be the Type_id column.

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO Student_sub_table.del OF DEL SELECT * FROM
   Regular_Student
DB2 CONNECT TO Target_db
DB2 IMPORT FROM Student_sub_table.del OF DEL METHOD P(1,2,3,5,4)
   MODIFIED BY NO_TYPE_ID INSERT INTO HIERARCHY (Student)
```

In this example, the target table STUDENT must exist. Since STUDENT is a subtable, the modifier `no_type_id` is used to indicate that there is no Type_id in the first column. However, you must ensure that there is an existing Object_id column, in addition to all of the other attributes that exist in the STUDENT table. Object-id is expected to be the first column in each row imported into the STUDENT table. The **METHOD** clause reverses the order of the last two attributes.

LBAC-protected data import considerations

For a successful import operation into a table with protected rows, you must have LBAC (label-based access control) credentials. You must also provide a valid security label, or a security label that can be converted to a valid label, for the security policy currently associated with the target table.

If you do not have valid LBAC credentials, the import fails and an error (SQLSTATE 42512) is returned. In cases where the input data does not contain a security label or that security label is not in its internal binary format, you can use several file type modifiers to allow your import to proceed.

When you import data into a table with protected rows, the target table has one column with a data type of DB2SECURITYLABEL. If the input row of data does not contain a value for that column, that row is rejected unless the `usedefaults` file type modifier is specified in the import command, in which case the security label you hold for write access from the security policy protecting the table is used. If you do not hold a security label for write access, the row is rejected and processing continues on to the next row.

When you import data into a table that has protected rows and the input data does include a value for the column with a data type of DB2SECURITYLABEL, the same rules are followed as when you insert data into that table. If the security label protecting the row being imported (the one in that row of the data file) is one that you are able to write to, then that security label is used to protect the row. (In other words, it is written to the column that has a data type of DB2SECURITYLABEL.) If you are not able to write to a row protected by that security label, what happens depends on how the security policy protecting the source table was created:

- If the CREATE SECURITY POLICY statement that created the policy included the option `RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL`, the insert fails and an error is returned.
- If the CREATE SECURITY POLICY statement did not include the option or if it instead included the `OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL` option, the security label in the data file for that row is ignored and the security label you hold for write access is used to protect that row. No error or warning is issued in this case. If you do not hold a security label for write access, the row is rejected and processing continues on to the next row.

Delimiter considerations

When importing data into a column with a data type of DB2SECURITYLABEL, the value in the data file is assumed by default to be the actual bytes that make up the internal representation of that security label. However, some raw data might contain newline characters which could be misinterpreted by the IMPORT command as delimiting the row. If you have this problem, use the `delprioritychar` file type modifier to ensure that the character delimiter takes precedence over the row delimiter. When you use `delprioritychar`, any record or column delimiters that are contained within character delimiters are not recognized as being delimiters. Using the `delprioritychar` file type modifier is safe to do even if none of the values contain a newline character, but it does slow the import down slightly.

If the data being imported is in ASC format, you might want to take an extra step in order to prevent any trailing white space from being included in the imported security labels and security label names. ASCII format uses column positions as delimiters, so this might occur when importing into variable-length fields. Use the `striptblanks` file type modifier to truncate any trailing blank spaces.

Nonstandard security label values

You can also import data files in which the values for the security labels are strings containing the values of the components in the security label, for example, *S:(ALPHA,BETA)*. To do so you must use the file type modifier `seclabelchar`. When you use `seclabelchar`, a value for a column with a data type of DB2SECURITYLABEL is assumed to be a string constant containing the security label in the string format for security labels. If a string is not in the proper format, the row is not inserted and a warning (SQLSTATE 01H53) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table, the row is not inserted and a warning (SQLSTATE 01H53) is returned.

You can also import a data file in which the values of the security label column are security label names. To import this sort of file you must use the file type modifier `seclabelname`. When you use `seclabelname`, all values for columns with a data type of DB2SECURITYLABEL are assumed to be string constants containing the names of existing security labels. If no security label exists with the indicated name for the security policy protecting the table, the row is not inserted and a warning (SQLSTATE 01H53) is returned.

Examples

For all examples, the input data file `myfile.del` is in DEL format. All are importing data into a table named REPS, which was created with this statement:

```
create table reps (row_label db2securitylabel,
id integer,
name char(30))
security policy data_access_policy
```

For this example, the input file is assumed to contain security labels in the default format:

```
db2 import from myfile.del of del modified by delprioritychar insert into reps
```

For this example, the input file is assumed to contain security labels in the security label string format:

```
db2 import from myfile.del of del modified by seclabelchar insert into reps
```

For this example, the input file is assumed to contain security labels names for the security label column:

```
db2 import from myfile.del of del modified by seclabelname insert into reps
```

Identity column import considerations

The import utility can be used to import data into a table containing an identity column whether or not the input data has identity column values.

If no identity-related file type modifiers are used, the utility works according to the following rules:

- If the identity column is GENERATED ALWAYS, an identity value is generated for a table row whenever the corresponding row in the input file is missing a value for the identity column, or a NULL value is explicitly given. If a non-NULL value is specified for the identity column, the row is rejected (SQL3550W).
- If the identity column is GENERATED BY DEFAULT, the import utility makes use of user-supplied values, if they are provided; if the data is missing or explicitly NULL, a value is generated.

The import utility does not perform any extra validation of user-supplied identity values beyond what is normally done for values of the identity column's data type (that is, SMALLINT, INT, BIGINT, or DECIMAL). Duplicate values will not be reported. In addition, the `compound=`*x* modifier cannot be used when importing data into a table with an identity column.

There are two ways you can simplify the import of data into tables that contain an identity column: the `identitymissing` and the `identityignore` file type modifiers.

Importing data without an identity column

The `identitymissing` modifier makes importing a table with an identity column more convenient if the input data file does not contain any values (not even NULLS) for the identity column. For example, consider a table defined with the following SQL statement:

```
create table table1 (c1 char(30),
                     c2 int generated by default as identity,
                     c3 real,
                     c4 char(1))
```

A user might want to import data from a file (`import.del`) into TABLE1, and this data might have been exported from a table that does not have an identity column. The following is an example of such a file:

```
Robert, 45.2, J
Mike, 76.9, K
Leo, 23.4, I
```

One way to import this file would be to explicitly list the columns to be imported through the **IMPORT** command as follows:

```
db2 import from import.del of del replace into table1 (c1, c3, c4)
```

For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of importing the file is to use the `identitymissing` file type modifier as follows:

```
db2 import from import.del of del modified by identitymissing
   replace into table1
```

Importing data with an identity column

The `identityignore` modifier is in some ways the opposite of the `identitymissing` modifier: it indicates to the import utility that even though the input data file contains data for the identity column, the data should be ignored, and an identity value should be generated for each row. For example, a user might want to import the following data from a file (`import.del`) into TABLE1, as defined previously:

```
Robert, 1, 45.2, J
Mike, 2, 76.9, K
Leo, 3, 23.4, I
```

If the user-supplied values of 1, 2, and 3 are not to be used for the identity column, the user could issue the following **IMPORT** command:

```
db2 import from import.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The identityignore modifier simplifies the syntax as follows:

```
db2 import from import.del of del modified by identityignore
   replace into table1
```

When a table with an identity column is exported to an IXF file, the REPLACE_CREATE and the CREATE options of the **IMPORT** command can be used to re-create the table, including its identity column properties. If such an IXF file is created from a table containing an identity column of type GENERATED ALWAYS, the only way that the data file can be successfully imported is to specify the identityignore modifier. Otherwise, all rows will be rejected (SQL3550W).

Note: The CREATE and REPLACE_CREATE options of the IMPORT command are deprecated and might be removed in a future release.

Generated column import considerations

The import utility can be used to import data into a table containing (nonidentity) generated columns whether or not the input data has generated column values.

If no generated column-related file type modifiers are used, the import utility works according to the following rules:

- A value is generated for a generated column whenever the corresponding row in the input file is missing a value for the column, or a NULL value is explicitly given. If a non-NULL value is supplied for a generated column, the row is rejected (SQL3550W).
- If the server generates a NULL value for a generated column that is not nullable, the row of data to which this field belongs is rejected (SQL0407N). This could happen, for example, if a non-nullable generated column were defined as the sum of two table columns that have NULL values supplied to them in the input file.

There are two ways you can simplify the import of data into tables that contain a generated column: the generatedmissing and the generatedignore file type modifiers.

Importing data without generated columns
The generatedmissing modifier makes importing data into a table with generated columns more convenient if the input data file does not contain any values (not even NULLS) for all generated columns present in the table. For example, consider a table defined with the following SQL statement:

```
create table table1 (c1 int,
                     c2 int,
                     g1 int generated always as (c1 + c2),
                     g2 int generated always as (2 * c1),
                     c3 char(1))
```

A user might want to import data from a file (load.del) into TABLE1, and this data might have been exported from a table that does not have any generated columns. The following is an example of such a file:

```
1, 5, J
2, 6, K
3, 7, I
```

One way to import this file would be to explicitly list the columns to be imported through the **IMPORT** command as follows:

```
db2 import from import.del of del replace into table1 (c1, c2, c3)
```

For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of importing the file is to use the generatedmissing file type modifier as follows:

```
db2 import from import.del of del modified by generatedmissing
   replace into table1
```

Importing data with generated columns

The generatedignore modifier is in some ways the opposite of the generatedmissing modifier: it indicates to the import utility that even though the input data file contains data for all generated columns, the data should be ignored, and values should be generated for each row. For example, a user might want to import the following data from a file (import.del) into TABLE1, as defined previously:

```
1, 5, 10, 15, J
2, 6, 11, 16, K
3, 7, 12, 17, I
```

The user-supplied, non-NULL values of 10, 11, and 12 (for g1), and 15, 16, and 17 (for g2) result in the row being rejected (SQL3550W). To avoid this, the user could issue the following **IMPORT** command:

```
db2 import from import.del of del method P(1, 2, 5)
   replace into table1 (c1, c2, c3)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The generatedignore modifier simplifies the syntax as follows:

```
db2 import from import.del of del modified by generatedignore
   replace into table1
```

For an INSERT_UPDATE, if the generated column is also a primary key and the generatedignore modifier is specified, the **IMPORT** command honors the generatedignore modifier. The **IMPORT** command does not substitute the user-supplied value for this column in the WHERE clause of the UPDATE statement.

LOB import considerations

Since the import utility restricts the size of a single column value to 32 KB, extra considerations need to be taken when importing LOBs.

The import utility, by default, treats data in the input file as data to load into the column. However, when large object (LOB) data is stored in the main input data file, the size of the data is limited to 32 KB. Therefore, to prevent loss of data, LOB data should be stored separate from the main datafile and the lobsinfile file type modifier should be specified when importing LOBs.

The LOBS FROM clause implicitly activates lobsinfile. The LOBS FROM clause conveys to the import utility the list of paths to search for the LOB files while importing the data. If LOBS FROM option is not specified, the LOB files to import are assumed to reside in the same path as the input relational data file.

Indicating where LOB data is stored

The LOB Location Specifier (LLS) can be used to store multiple LOBs in a single file when importing the LOB information. The export utility generates and stores it in the export output file when `lobsinfile` is specified, and it indicates where LOB data can be found. When data with the modified by `lobsinfile` option specified is being imported, the database will expect an LLS for each of the corresponding LOB columns. If something other than an LLS is encountered for a LOB column, the database will treat it as a LOB file and will load the entire file as the LOB.

For an import in `CREATE` mode, you can specify that the LOB data be created and stored in a separate table space by using the LONG IN clause.

The following example shows how you would import an DEL file which has its LOBs stored in separate files:

```
IMPORT FROM inputfile.del OF DEL
 LOBS FROM /tmp/data
 MODIFIED BY lobsinfile
 INSERT INTO newtable
```

User-defined distinct types import considerations

The import utility casts user-defined distinct types (UDTs) to similar base data types automatically. This saves you from having to explicitly cast UDTs to the base data types. Casting allows for comparisons between UDTs and the base data types in SQL.

Client/server environments and import

When you import a file to a remote database, a stored procedure can be called to perform the import on the server.

A stored procedure cannot be called when:

- The application and database code pages are different.
- The file being imported is a multiple-part PC/IXF file.
- The method used for importing the data is either column name or relative column position.
- The target column list provided is longer than 4 KB.
- The LOBS FROM clause or the `lobsinfile` modifier is specified.
- The NULL INDICATORS clause is specified for ASC files.

When import uses a stored procedure, messages are created in the message file using the default language installed on the server. The messages are in the language of the application if the language at the client and the server are the same.

The import utility creates two temporary files in the `tmp` subdirectory of the `sqllib` directory (or the directory indicated by the **DB2INSTPROF** registry variable, if specified). One file is for data, and the other file is for messages generated by the import utility.

If you receive an error about writing or opening data on the server, ensure that:

- The directory exists.
- There is sufficient disk space for the files.
- The instance owner has write permission in the directory.

Table locking modes supported by the import utility

The import utility supports two table locking modes: offline, or ALLOW NO ACCESS, mode; and online, or ALLOW WRITE ACCESS mode.

ALLOW NO ACCESS mode prevents concurrent applications from accessing table data. ALLOW WRITE ACCESS mode allows concurrent applications both read and write access to the import target table. If no mode is explicitly specified, import runs in the default mode, ALLOW NO ACCESS. As well, the import utility is, by default, bound to the database with isolation level RS (read stability).

Offline import (ALLOW NO ACCESS)

In ALLOW NO ACCESS mode, import acquires an exclusive (X) lock on the target table is before inserting any rows. Holding a lock on a table has two implications:

- First, if there are other applications holding a table lock or row locks on the import target table, the import utility waits for those applications to commit or roll back their changes.
- Second, while import is running, any other application requesting locks waits for the import operation to complete.

Note: You can specify a locktimeout value, which prevents applications (including the import utility) from waiting indefinitely for a lock.
By requesting an exclusive lock at the beginning of the operation, import prevents deadlocks from occurring as a result of other applications working and holding row locks on the same target table.

Online import (ALLOW WRITE ACCESS)

In ALLOW WRITE ACCESS mode, the import utility acquires a nonexclusive (IX) lock on the target table. Holding this lock on the table has the following implications:

- If there are other applications holding an incompatible table lock, the import utility does not start inserting data until all of these applications commit or roll back their changes.
- While import is running, any other application requesting an incompatible table lock waits until the import commits or rolls back the current transaction. Note that import's table lock does not persist across a transaction boundary. As a result, online import has to request and potentially wait for a table lock after every commit.
- If there are other applications holding an incompatible row lock, the import utility stops inserting data until all of these applications commit or roll back their changes.
- While import is running, any other application requesting an incompatible row lock waits until the import operation commits or rolls back the current transaction.

To preserve the online properties, and to reduce the chance of a deadlock, an ALLOW WRITE ACCESS import periodically commits the current transaction and releases all row locks before escalating to an exclusive table lock. If you have not explicitly set a commit frequency, import performs commits as if COMMITCOUNT AUTOMATIC has been specified. No commits are performed if COMMITCOUNT is set to 0.

ALLOW WRITE ACCESS mode is not compatible with the following:

- Imports in REPLACE, CREATE, or REPLACE_CREATE mode

- Imports with buffered inserts
- Imports into a target view
- Imports into a hierarchy table
- Imports into a table with its lock granularity is set at the table level (set by using the LOCKSIZE parameter of the ALTER TABLE statement)

Importing XML data

The import utility can be used to import XML data into an XML table column using either the table name or a nickname for a DB2 for Linux, UNIX, and Windows source data object.

When importing data into an XML table column, you can use the XML FROM option to specify the paths of the input XML data file or files. For example, for an XML file "/home/user/xmlpath/xmldocs.001.xml" that had previously been exported, the following command could be used to import the data back into the table.

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

Validating inserted documents against schemas

The XMLVALIDATE option allows XML documents to be validated against XML schemas as they are imported. In the following example, incoming XML documents are validated against schema information that was saved when the XML documents were exported:

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T1
```

Specifying parse options

You can use the XMLPARSE option to specify whether whitespace in the imported XML documents is preserved or stripped. In the following example, all imported XML documents are validated against XML schema information that was saved when the XML documents were exported, and these documents are parsed with whitespace preserved.

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING XDS INSERT INTO USER.T1
```

Chapter 36. Export utility

The export utility extracts data using an SQL select or an XQuery statement, and places that information into a file. You can use the output file to move data for a future import or load operation or to make the data accessible for analysis.

The export utility is a relatively simple, yet flexible data movement utility. You can activate it by issuing the **EXPORT** command in the CLP, by calling the ADMIN_CMD stored procedure, or by calling the db2Export API through a user application.

The following items are mandatory for a basic export operation:

- The path and name of the operating system file in which you want to store the exported data
- The format of the data in the input file
  Export supports IXF and DEL data formats for the output files.
- A specification of the data that is to be exported
  For the majority of export operations, you need to provide a SELECT statement that specifies the data to be retrieved for export. When exporting typed tables, you don't need to issue the SELECT statement explicitly; you only need to specify the subtable traverse order within the hierarchy

You can use the export utility with DB2 Connect if you need to move data in IXF format.

Additional options

There are a number of parameters that allow you to customize an export operation. File type modifiers offer many options such as allowing you to change the format of the data, date and time stamps, or code page, or have certain data types written to separate files. Using the **METHOD** parameters, you can specify different column names to be used for the exported data.

You can export from tables that include one or more columns with an XML data type. Use the **XMLFILE**, **XML TO**, and **XMLSAVESCHEMA** parameters to specify details about how those exported documents are stored.

There are a few ways to improve the export utility's performance. As the export utility is an embedded SQL application and does SQL fetches internally, optimizations that apply to SQL operations apply to the export utility as well. Consider taking advantage of large buffer pools, indexing, and sort heaps. In addition, try to minimize device contention on the output files by placing them away from the containers and log devices.

The messages file

The export utility writes error, warning, and informational messages to standard ASCII text message files. For all interfaces except the CLP, you must specify the name of these files in advance with the **MESSAGES** parameter. If you are using the CLP and do not specify a messages file, the export utility writes the messages to standard output.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for exporting data. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Privileges and authorities required to use the export utility

Privileges enable you to create, update, delete, or access database resources. Authority levels provide a method of mapping privileges to higher-level database manager maintenance and utility operations.

Together, privileges and authorities control access to the database manager and its database objects. You can access only those objects for which you have the appropriate authorization: that is, the required privilege or authority.

You must have DATAACCESS authority or the CONTROL or SELECT privilege for each table or view participating in the export operation.

When you are exporting LBAC-protected data, the session authorization ID must be allowed to read the rows or columns that you are trying to export. Protected rows that the session authorization ID is not authorized to read are not exported. If the SELECT statement includes any protected columns that the session authorization ID is not allowed to read, the export utility fails, and an error (SQLSTATE 42512) is returned.

Exporting data

Use the export utility to export data from a database to a file. The file can have one of several external file formats. You can specify the data to be exported by supplying an SQL SELECT statement or by providing hierarchical information for typed tables.

Before you begin

You need DATAACCESS authority, the CONTROL privilege, or the SELECT privilege on each participating table or view to export data from a database

Before running the export utility, you must be connected (or be able to implicitly connect) to the database from which you want to export the data. If implicit connect is enabled, a connection to the default database is established. Utility access to Linux, UNIX, or Windows database servers from Linux, UNIX, or Windows clients must be through a direct connection through the engine and not through a DB2 Connect gateway or loop back environment.

Because the utility issues a COMMIT statement, complete all transactions and release all locks by issuing a COMMIT or a ROLLBACK statement before running the export utility. There is no requirement for applications accessing the table and using separate connections to disconnect.

You cannot export tables with structured type columns.

Procedure

To run the export utility:

- Specify the **EXPORT** command in the command line processor (CLP).
- Call the db2Export application programming interface (API).
- Open the task assistant in IBM Data Studio for the **EXPORT** command.

Example

A simple export operation requires you to specify only a target file, a file format, and a source file for the SELECT statement.

For example:

```
db2 export to filename of ixf select * from table
```

where *filename* is the name of the output file that you want to create and export, ixf is the file format, and *table* is the name of the table that contains the data you want to copy.

However, you might also want to specify a messages file to which warning and error messages are written. To do that, add the **MESSAGES** parameter and a message file name (in this case, msg.txt). For example:

```
db2 export to filename of ixf messages msgs.txt select * from table
```

Export sessions - CLP examples

Example 1

The following example shows how to export information from the STAFF table in the SAMPLE database (to which the user must be connected) to myfile.ixf, with the output in IXF format. If the database connection is not through DB2 Connect, the index definitions (if any) will be stored in the output file; otherwise, only the data will be stored:

```
db2 export to myfile.ixf of ixf messages msgs.txt select * from staff
```

Example 2

The following example shows how to export the information about employees in Department 20 from the STAFF table in the SAMPLE database (to which the user must be connected) to awards.ixf, with the output in IXF format:

```
db2 export to awards.ixf of ixf messages msgs.txt select * from staff
   where dept = 20
```

Example 3

The following example shows how to export LOBs to a DEL file:

```
db2 export to myfile.del of del lobs to mylobs/
   lobfile lobs1, lobs2 modified by lobsinfile
   select * from emp_photo
```

Example 4

The following example shows how to export LOBs to a DEL file, specifying a second directory for files that might not fit into the first directory:

```
db2 export to myfile.del of del
   lobs to /db2exp1/, /db2exp2/ modified by lobsinfile
   select * from emp_photo
```

Example 5

The following example shows how to export data to a DEL file, using a single quotation mark as the string delimiter, a semicolon as the column delimiter, and a comma as the decimal point. The same convention should be used when importing data back into the database:

```
db2 export to myfile.del of del
   modified by chardel'' coldel; decpt,
   select * from staff
```

LBAC-protected data export considerations

When you export data that is protected by label-based access control (LBAC), the data that is exported is limited to the data that your LBAC credentials allow you to read.

If your LBAC credentials do not allow you to read a row, that row is not exported, but no error is returned. If your LBAC credentials do not allow you to read a column, the export utility fails, and an error (SQLSTATE 42512) is returned.

A value from a column with a data type of DB2SECURITYLABEL is exported as raw data enclosed in character delimiters. If a character delimiter is included in the original data, it is doubled. No other changes are made to the bytes that make up the exported value. This means that a data file that contains DB2SECURITYLABEL data can contain newlines, formfeeds, or other non-printable ASCII characters.

If you want the values of columns with a data type of DB2SECURITYLABEL to be exported in a human-readable form, you can use the SECLABEL_TO_CHAR scalar function in the SELECT statement to convert the values to the security label string format.

Examples

In the following examples, output is in DEL format and is written to the file `myfile.del`. The data is exported from a table named REPS, which was created with the following statement:

```
create table reps (row_label db2securitylabel,
id integer,
name char(30))
security policy data_access_policy
```

This example exports the values of the row_label column in the default format:

```
db2 export to myfile.del of del select * from reps
```

The data file is not very readable in most text editors because the values for the row_label column are likely to contain several ASCII control characters.

The following example exports the values of the row_label column in the security label string format:

```
db2 export to myfile.del of del select SECLABEL_TO_CHAR
(row_label,'DATA_ACCESS_POLICY'), id, name from reps
```

Here is an excerpt of the data file created by the previous example. Notice that the format of the security label is readable:

```
...
"Secret:():Epsilon 37", 2005, "Susan Liu"
"Secret:():(Epsilon 37,Megaphone,Cloverleaf)", 2006, "Johnny Cogent"
"Secret:():(Megaphone,Cloverleaf)", 2007, "Ron Imron"
...
```

Table export considerations

A typical export operation involves the outputting of selected data that is inserted or loaded into existing tables. However, it is also possible to export an entire table for subsequent re-creation using the import utility.

To export a table, you must specify the PC/IXF file format. You can then re-create your saved table (including its indexes) using the import utility in CREATE mode. However, some information is not saved to the exported IXF file if any of the following conditions exist:

- The index column names contain hexadecimal values of 0x2B or 0x2D.
- The table contains XML columns.
- The table is multidimensional clustered (MDC).
- The table contains a table partitioning key.
- The index name is longer than 128 bytes due to code page conversion.
- The table is protected.
- The **EXPORT** command contains action strings other than SELECT * FROM *tablename*
- You specify the **METHOD N** parameter for the export utility.

For a list of table attributes that are lost, see "Table import considerations." If any information is not saved, warning SQL27984W is returned when the table is re-created.

Note: Import's CREATE mode is being deprecated. Use the **db2look** utility to capture and re-create your tables.

Index information
If the column names specified in the index contain either - or + characters, the index information is not collected, and warning SQL27984W is returned. The export utility completes its processing, and the data exported is unaffected. However, the index information is not saved in the IXF file. As a result, you must create the indexes separately using the **db2look** utility.

Space limitations
The export operation fails if the data that you are exporting exceeds the space available on the file system on which the exported file is created. In this case, you should limit the amount of data selected by specifying conditions on the WHERE clause so that the exported file fits on the target file system. You can run the export utility multiple times to export all of the data.

Tables with other file formats
If you do not export using the IXF file format, the output files do not contain descriptions of the target table, but they contain the record data. To re-create a table and its data, create the target table, then use the load or import utility to populate the table. You can use the **db2look** utility to capture the original table definitions and to generate the corresponding data definition language (DDL).

Typed table export considerations

You can use the DB2 export utility can be used to move data out of typed tables for a later import. Export moves data from one hierarchical structure of typed tables to another by following a specific order and creating an intermediate flat file.

When working with typed tables, the export utility controls what is placed in the output file; specify only the target table name and, optionally, the WHERE clause.

You can express subselect statements only by specifying the target table name and the WHERE clause. You cannot specify a fullselect or select-statement when exporting a hierarchy.

Preservation of hierarchies using traverse order

Typed tables can be in a hierarchy. There are several ways you can move data across hierarchies:

- Movement from one hierarchy to an identical hierarchy
- Movement from one hierarchy to a subsection of a larger hierarchy
- Movement from a subsection of a large hierarchy to a separate hierarchy

Identification of types in a hierarchy is database dependent, meaning that in different databases, the same type has a different identifier. Therefore, when moving data between these databases, a mapping of the same types must be done to ensure that the data is moved correctly.

The mapping used for typed tables is known as the *traverse order*, the order of proceeding top-to-bottom, left-to-right through all of the supertables and subtables in the hierarchy. Before each typed row is written out during an export operation, an identifier is translated into an index value. This index value can be any number from one to the number of relevant types in the hierarchy. Index values are generated by numbering each type when moving through the hierarchy in a specific order-the traverse order. Figure 1 shows a hierarchy with four valid traverse orders:

- Person, Employee, Manager, Architect, Student
- Person, Student, Employee, Manager, Architect
- Person, Employee, Architect, Manager, Student
- Person, Student, Employee, Architect, Manager

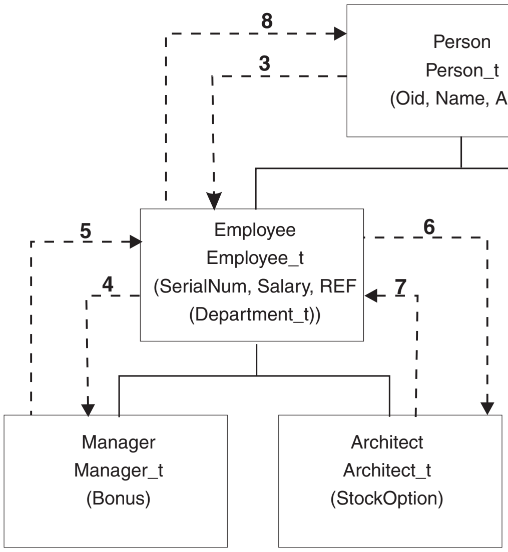


Figure 51. An example of a hierarchy

The traverse order is important when moving data between table hierarchies because it determines where the data is moved in relation to other data. There are two types of traverse order: *default* and *user specified*.

Default traverse order

With the default traverse order, all relevant types refer to all reachable types in the hierarchy from a given starting point in the hierarchy. The default order includes all tables in the hierarchy, and each table is ordered by the scheme used in the OUTER order predicate. For instance, the default traverse order of Figure 1, indicated by the dotted line, would be Person, Student, Employee, Manager, Architect.

The default traverse order behaves differently when used with different file formats. Exporting data to the PC/IXF file format creates a record of all relevant types, their definitions, and relevant tables. The export utility also completes the mapping of an index value to each table. When working with the PC/IXF file format, you should use the default traverse order.

With the ASC or DEL file format, the order in which the typed rows and the typed tables are created could be different, even though the source and target hierarchies might be structurally identical. This results in time differences that the default traverse order identifies when proceeding through the hierarchies. The creation time of each type determines the order used to move through the hierarchy at both the source and the target when using the default traverse order. Ensure that the creation order of each type in both the source and the target hierarchies is identical and that there is structural identity between the source and the target. If these conditions cannot be met, select a user-specified traverse order.

User-specified traverse order

With the user-specified traverse order, you define (in a traverse order list) the relevant types to be used. This order outlines how to traverse the hierarchy and what sub-tables to export, whereas with the default traverse order, all tables in the hierarchy are exported.

Although you determine the starting point and the path down the hierarchy when defining the traverse order, remember that the subtables must be traversed in *pre-order* fashion. Each branch in the hierarchy must be traversed to the bottom before a new branch can be started. The export utility looks for violations of this condition within the specified traverse order. One method of ensuring that the condition is met is to proceed from the top of the hierarchy (or the root table), down the hierarchy (subtables) to the bottom subtable, then back up to its supertable, down to the next "right-most" subtable, then back up to next higher supertable, down to its subtables, and so on.

If you want to control the traverse order through the hierarchies, ensure that the same traverse order is used for both the export and the import utilities.

Example 1

The following examples are based on the hierarchical structure in Figure 1. To export the entire hierarchy, enter the following commands:

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO entire_hierarchy.ixf OF IXF HIERARCHY STARTING Person
```

Note that setting the parameter **HIERARCHY STARTING** to `Person` indicates that the default traverse order starting from the table PERSON.

Example 2

To export the entire hierarchy, but only the data for those people over the age of 20, you would enter the following commands:

```
DB2 CONNECT TO Source_db
   DB2 EXPORT TO entire_hierarchy.del OF DEL HIERARCHY (Person,
      Employee, Manager, Architect, Student) WHERE Age>=20
```

Note that setting the parameter **HIERARCHY** to Person, Employee, Manager, Architect, Student indicates a user-specified traverse order.

Identity column export considerations

You can use the export utility to export data from a table containing an identity column. However, the identity column limits your choice of output file format.

If the SELECT statement that you specify for the export operation is of the form SELECT * FROM *tablename* and you do not use the METHOD option, exporting identity column properties to IXF files is supported. You can then use the REPLACE_CREATE and the CREATE options of the **IMPORT** command to re-create the table, including its identity column properties. If you create the exported IXF file from a table containing an identity column of type GENERATED ALWAYS, the only way that you can successfully import the data file is to specify the identityignore file type modifier during the import operation. Otherwise, all rows are rejected (SQL3550W is issued).

Note: The CREATE and REPLACE_CREATE options of the IMPORT command are deprecated and might be removed in a future release.

LOB export considerations

When exporting tables with large object (LOB) columns, the default action is to export a maximum of 32 KB per LOB value and to place it in the same file as the rest of the column data. If you are exporting LOB values that exceed 32 KB, you should have the LOB data written to a separate file to avoid truncation.

To specify that LOB should be written to its own file, use the lobsinfile file type modifier. This modifier instructs the export utility to place the LOB data in the directories specified by the LOBS TO clause. Using LOBS TO or LOBFILE implicitly activates the lobsinfile file type modifier. By default, LOB values are written to the same path to which the exported relational data is written. If one or more paths are specified with the LOBS TO option, the export utility cycles between the paths to write each successful LOB value to the appropriate LOB file. You can also specify names for the output LOB files using the LOBFILE option. If the LOBFILE option is specified, the format of lobfilename is lobfilespec.xxx.lob, where lobfilespec is the value specified for the LOBFILE option, and xxx is a sequence number for LOB files produced by the export utility. Otherwise, lobfilename is of the format: exportfilename.xxx.lob, where exportfilename is the name of the exported output file specified for the EXPORT command, and xxx is a sequence number for LOB files produced by the export utility.

By default, LOBs are written to a single file, but you can also specify that the individual LOBs are to be stored in separate files. The export utility generates a LOB Location Specifier (LLS) to enable the storage of multiple LOBs in one file. The LLS, which is written to the export output file, is a string that indicates where the LOB data is stored within the file. The format of the LLS is lobfilename.ext.nnn.mmm/, where lobfilename.ext is the name of the file that contains the LOB, nnn is the offset of the LOB within the file (measured in bytes), and mmm is the length of the LOB (measured in bytes). For example, an LLS of db2exp.001.123.456/ indicates that the LOB is located in the file db2exp.001, begins at an offset of 123 bytes into the file, and is 456 bytes long. If the indicated size in the LLS is 0, the LOB is considered to have a length of 0. If the length is -1, the LOB is considered to be NULL and the offset and file name are ignored.

If you don't want individual LOB data concatenated to the same file, use the lobsinsepfiles file type modifier to write each LOB to a separate file.

Note: The IXF file format does not store the LOB options of the column, such as whether or not the LOB column is logged. This means that the import utility cannot re-create a table containing a LOB column that is defined to be 1 GB or larger.

Example 1
The following example shows how to export LOBs (where the exported LOB files have the specified base name lobs1) to a DEL file:

```
db2 export to myfile.del of del lobs to mylobs/
   lobfile lobs1 modified by lobsinfile
   select * from emp_photo
```

Example 2
The following example shows how to export LOBs to a DEL file, where each LOB value is written to a separate file and lobfiles are written to two directories:

```
db2 export to myfile.del of del
lobs to /db2exp1/, /db2exp2/ modified by lobsinfile
select * from emp_photo
```

Exporting XML data

When exporting XML data, the resulting QDM (XQuery Data Model) instances are written to a file or files separate from the main data file containing exported relational data. This is true even if neither the XMLFILE nor the XML TO option is specified.

By default, exported QDM instances are all concatenated to the same XML file. You can use the XMLINSEPFILES file type modifier to specify that each QDM instance be written to a separate file.

The XML data, however, is represented in the main data file with an XML data specifier (XDS). The XDS is a string represented as an XML tag named "XDS", which has attributes that describe information about the actual XML data in the column; such information includes the name of the file that contains the actual XML data, and the offset and length of the XML data within that file.

The destination paths and base names of the exported XML files can be specified with the XML TO and XMLFILE options. If the XML TO or XMLFILE option is specified, the format of the exported XML file names, stored in the FIL attribute of the XDS, is xmlfilespec.xxx.xml, where xmlfilespec is the value specified for the XMLFILE option, and xxx is a sequence number for xml files produced by the export utility. Otherwise, the format of the exported XML file names is: exportfilename.xxx.xml, where exportfilename is the name of the exported output file specified for the EXPORT command, and xxx is a sequence number for xml files produced by the export utility.

By default, exported XML files are written to the path of the exported data file. The default base name for exported XML files is the name of the exported data file, with an appending 3-digit sequence number, and the `.xml` extension.

Examples

For the following examples, imagine a table USER.T1 containing four columns and two rows:

```
C1 INTEGER
C2 XML
C3 VARCHAR(10)
C4 XML
```

Table 112. USER.T1

C1	C2	C3	C4
2	<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to><from> Me</from><heading>note1</heading><body>Hello World!</body></note>	'char1'	<?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him</to><from> Her</from><heading>note2</heading><body>Hello World!</body></note>
4	NULL	'char2'	?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">to>Us</to><from> Them</from><heading>note3</heading><body>Hello World!</body></note>

Example 1

The following command exports the contents of USER.T1 in Delimited ASCII (DEL) format to the file "/mypath/t1export.del". Because the XML TO and XMLFILE options are not specified, the XML documents contained in columns C2 and C4 are written to the same path as the main exported file "/mypath". The base name for these files is "t1export.del.xml". The XMLSAVESCHEMA option indicates that XML schema information is saved during the export procedure.

```
EXPORT TO /mypath/t1export.del OF DEL XMLSAVESCHEMA SELECT * FROM USER.T1
```

The exported file "/mypath/t1export.del" contains:

```
2,"<XDS FIL='t1export.del.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='t1export.del.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='t1export.del.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

The exported XML file "/mypath/t1export.del.001.xml" contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him
   </to><from>Her</from><heading>note2</heading><body>Hello World!
   </body></note><?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">
   <to>Us</to><from>Them</from>heading>note3</heading><body>
   Hello World!</body></note>
```

Example 2

The following command exports the contents of USER.T1 in DEL format to the file "t1export.del". XML documents contained in columns C2 and C4 are written to the path "/home/user/xmlpath". The XML files are named with the base name "xmldocs", with multiple exported XML documents written to the same XML file. The XMLSAVESCHEMA option indicates that XML schema information is saved during the export procedure.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs XMLSAVESCHEMA SELECT * FROM USER.T1
```

The exported DEL file "/home/user/t1export.del" contains:

```
2,"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='xmldocs.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='xmldocs.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

The exported XML file "/home/user/xmlpath/xmldocs.001.xml" contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00">
   <to>Him</to><from>Her</from><heading>note2</heading><body>
   Hello World!</body></note><?xml version="1.0" encoding="UTF-8" ?>
   <note time="14:00:00"><to>Us</to><from>Them</from><heading>
   note3</heading><body>Hello World!</body></note>
```

Example 3

The following command is similar to Example 2, except that each exported XML document is written to a separate XML file.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLINSEPFILES XMLSAVESCHEMA
SELECT * FROM USER.T1
```

The exported file "/mypath/t1export.del" contains:

```
2,"<XDS FIL='xmldocs.001.xml' />","char1","XDS FIL='xmldocs.002.xml' />"
4,,"char2","<XDS FIL='xmldocs.004.xml' SCH='S1.SCHEMA_A' />"
```

The exported XML file "/home/user/xmlpath/xmldocs.001.xml" contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note>
```

The exported XML file "/home/user/xmlpath/xmldocs.002.xml" contains:

```
?xml version="1.0" encoding="UTF-8" ?>note time="13:00:00">to>Him/to>
   from>Her/from>heading>note2/heading>body>Hello World!/body>
   /note>
```

The exported XML file "/home/user/xmlpath/xmldocs.004.xml" contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00"><to>Us</to>
   <from>Them</from><heading>note3</heading><body>Hello World!</body>
   </note>
```

Example 4

The following command writes the result of an XQuery to an XML file.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLNODECLARATION select
xmlquery( '$m/note/from/text()' passing by ref c4 as "m" returning sequence)
   from USER.T1
```

The exported DEL file "/mypath/t1export.del" contains:

```
"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='3' />"
"<XDS FIL='xmldocs.001.xml' OFF='3' LEN='4' />"
```

The exported XML file "/home/user/xmlpath/xmldocs.001.xml" contains:

```
HerThem
```

Note: The result of this particular XQuery does not produce well-formed XML documents. Therefore, the file exported in this example, could not be directly imported into an XML column.

Chapter 37. Comparison between the ingest, import, and load utilities

The following tables summarize some of the key similarities and differences between the ingest, import, and load utilities.

Table 113. Supported table types

Table type	Ingest	Load	Import
Detached table	not supported	not supported	not supported
Global temporary table	not supported	not supported	not supported
Multidimensional clustering (MDC) or insert time clustering (ITC) table	supported	supported	supported
Materialized query table (MQT) that is maintained by user	supported	supported	supported
Nickname	supported	not supported	supported
Range-clustered table (RCT)	supported	not supported	supported
Range-partitioned table	supported	supported	supported
Summary table	supported	supported	supported
Temporal table	supported	supported	supported
Typed table	not supported	not supported	supported
Untyped (regular) table	supported	supported	supported
Updatable view (except typed view)	supported	not supported	supported

Table 114. Supported data types

Table type	Ingest	Load	Import
Numeric: SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE, DECFLOAT	supported	supported	supported
Character: CHAR, VARCHAR, NCHAR, NVARCHAR, plus corresponding FOR BIT DATA types	supported	supported	supported
Graphic: GRAPHIC, VARGRAPHIC	supported	supported	supported
Long types: LONG VARCHAR, LONG VARGRAPHIC	supported	supported	supported
Date/time: DATE, TIME, TIMESTAMP, including TIMESTAMP(p)	supported	supported	supported
DB2SECURITYLABEL	supported	supported	supported

Table 114. Supported data types (continued)

Table type	Ingest	Load	Import
LOBs from files: BLOB, CLOB, DBCLOB, NCLOB	not supported	supported	supported
Inline LOBs	not supported	supported	supported
XML from files	not supported	supported	supported
Inline XML	not supported	supported	supported
Distinct type	supported (if based on a supported built-in data type)	supported	supported
Structured type	not supported	not supported	supported
Reference type	supported	supported	supported

Table 115. Supported input sources

Input type	Ingest Restartable?	Load Restartable?	Import Restartable?
Cursor	not supported n/a	supported yes	not supported n/a
Device	not supported n/a	supported yes	not supported n/a
File	supported yes	supported yes	supported yes
Pipe	supported yes	supported yes	not supported n/a

Table 116. Supported input formats

Table type	Ingest	Load	Import
ASC (including binary)	supported	supported	supported
DB2 z/OS UNLOAD format	not supported	not supported	not supported
DEL	supported	supported	supported
IXF	not supported	supported	supported

There are a number of other important differences that distinguish the ingest utility from the load and import utility:

- The ingest utility allows the input records to contain extra fields between the fields that correspond to columns.
- The ingest utility supports update, delete, and merge.
- The ingest utility supports constructing column values from expressions containing field values.
- The ingest utility allows other applications to update the target table while ingest is running.

Chapter 38. Additional DB2 resources for data movement

The db2move command and the ADMIN_COPY_SCHEMA procedure facilitate the movement of large numbers of tables or entire schemas between table spaces or databases. You can also use the db2move command to re-create entire databases. The ADMIN_MOVE_TABLE stored procedure moves the data in an active table into a new table object with the same name, while the data remains online and available for access.

Copying schemas

The **db2move** utility and the ADMIN_COPY_SCHEMA procedure allow you to quickly make copies of a database schema. Once a model schema is established, you can use it as a template for creating new versions.

Procedure

- Use the ADMIN_COPY_SCHEMA procedure to copy a single schema within the same database.
- Use the **db2move** utility with the **-co COPY** action to copy a single schema or multiple schemas from a source database to a target database. Most database objects from the source schema are copied to the target database under the new schema.

Troubleshooting tips

Both the ADMIN_COPY_SCHEMA procedure and the **db2move** utility invoke the **LOAD** command. While the load is processing, the table spaces wherein the database target objects reside are put into backup pending state.

ADMIN_COPY_SCHEMA procedure

Using this procedure with the COPYNO option places the table spaces wherein the target object resides into backup pending state, as described in the previous note. To get the table space out of the set integrity pending state, this procedure issues a SET INTEGRITY statement. In situations where a target table object has referential constraints defined, the target table is also placed in the set integrity pending state. Because the table spaces are already in backup pending state, the attempt by the ADMIN_COPY_SCHEMA procedure to issue a SET INTEGRITY statement fails.

To resolve this situation, issue a **BACKUP DATABASE** command to get the affected table spaces out of backup pending state. Next, look at the `Statement_text` column of the error table generated by this procedure to find a list of tables in the set integrity pending state. Then issue the SET INTEGRITY statement for each of the tables listed to take each table out of the set integrity pending state.

db2move utility

This utility attempts to copy all allowable schema objects except for the following types:

- table hierarchy
- staging tables (not supported by the load utility in multiple partition database environments)

- jars (Java routine archives)
- nicknames
- packages
- view hierarchies
- object privileges (All new objects are created with default authorizations)
- statistics (New objects do not contain statistics information)
- index extensions (user-defined structured type related)
- user-defined structured types and their transform functions

Unsupported type errors

If an object of one of the unsupported types is detected in the source schema, an entry is logged to an error file. The error file indicates that an unsupported object type is detected. The COPY operation still succeeds; the logged entry is meant to inform you of objects not copied by this operation.

Objects not coupled with schemas

Objects that are not coupled with a schema, such as table spaces and event monitors, are not operated on during a copy schema operation. You should create them on the target database before the copy schema operation is invoked.

Replicated tables

When copying a replicated table, the new copy of the table is not enabled for replication. The table is recreated as a regular table.

Different instances

The source database must be cataloged if it does not reside in the same instance as the target database.

SCHEMA_MAP option

When using the SCHEMA_MAP option to specify a different schema name on the target database, the copy schema operation will perform only minimal parsing of the object definition statements to replace the original schema name with the new schema name. For example, any instances of the original schema that appear inside the contents of an SQL procedure are not replaced with the new schema name. Thus the copy schema operation might fail to recreate these objects. Other examples might include staging table, result table, materialized query table. You can use the DDL in the error file to manually recreate these failed objects after the copy operation completes.

Interdependencies between objects

The copy schema operation attempts to recreate objects in an order that satisfies the interdependencies between these objects. For example, if a table T1 contains a column that references a user-defined function U1, then it will recreate U1 before recreating T1. However, dependency information for procedures is not readily available in the catalogs, so when re-creating procedures, the copy schema operation will first attempt to re-create all procedures, then try to re-create those that failed again (on the assumption that if they depended on a procedure that was successfully created during the previous attempt, then during a subsequent attempt they will be re-created successfully). The operation will continually try to recreate these failed procedures as long as it is able to successfully recreate one or more during a subsequent attempt. During every attempt at recreating a procedure, an error (and DDL) is logged into the error file. You might see many entries in the error file for the same procedures, but these procedures might have even been successfully recreated during a subsequent attempt. You should query the SYSCAT.PROCEDURES table upon completion of the copy schema operation to determine if these procedures listed in the error file were successfully recreated.

For more information, see the ADMIN_COPY_SCHEMA procedure and the **db2move** utility.

Example of schema copy using the ADMIN_COPY_SCHEMA procedure

Use the ADMIN_COPY SCHEMA procedure as shown in the following example, to copy a single schema within the same database.

```
DB2 "SELECT SUBSTR(OBJECT_SCHEMA,1, 8)
  AS OBJECT_SCHEMA, SUBSTR(OBJECT_NAME,1, 15)
  AS OBJECT_NAME, SQLCODE, SQLSTATE, ERROR_TIMESTAMP, SUBSTR(DIAGTEXT,1, 80)
  AS DIAGTEXT, SUBSTR(STATEMENT,1, 80)
  AS STATEMENT FROM COPYERRSCH.COPYERRTAB"

CALL SYSPROC.ADMIN_COPY_SCHEMA('SOURCE_SCHEMA', 'TARGET_SCHEMA',
  'COPY', NULL, 'SOURCETS1 , SOURCETS2', 'TARGETTS1, TARGETTS2,
  SYS_ANY', 'ERRORSCHEMA', 'ERRORNAME')
```

The output from this SELECT statement is shown in the following example:

```
OBJECT_SCHEMA OBJECT_NAME     SQLCODE     SQLSTATE ERROR_TIMESTAMP
------------- --------------- ----------- -------- --------------------------
SALES         EXPLAIN_STREAM         -290 55039    2006-03-18-03.22.34.810346

DIAGTEXT
--------------------------------------------------------------------------------
[IBM][CLI Driver][DB2/LINUXX8664] SQL0290N  Table space access is not allowed.

STATEMENT
--------------------------------------------------------------------------------
set integrity for "SALES  "."ADVISE_INDEX" , "SALES"."ADVISE_MQT" , "SALES"."

1 record(s) selected.
```

Examples of schema copy by using the db2move utility

Use the **db2move** utility with the **-co COPY** action to copy one or more schemas from a source database to a target database. After a model schema is established, you can use it as a template for creating new versions.

Example 1: Using the -c COPY options

The following example of the **db2move -co COPY** options copies the schema BAR and renames it FOO from the sample database to the target database:

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" -u userid -p password
```

The new (target) schema objects are created by using the same object names as the objects in the source schema, but with the target schema qualifier. It is possible to create copies of tables with or without the data from the source table. The source and target databases can be on different systems.

Example 2: Specifying table space name mappings during the COPY operation

The following example shows how to specify specific table space name mappings to be used instead of the table spaces from the source system during a **db2move COPY** operation. You can specify the SYS_ANY keyword to indicate that the target table space must be chosen by using the default table space selection algorithm. In this case, the **db2move** utility chooses any available table space to be used as the target:

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" tablespace_map "(SYS_ANY)" -u userid -p password
```

The SYS_ANY keyword can be used for all table spaces, or you can specify specific mappings for some table spaces, and the default table space selection algorithm for the remaining:

```
db2move sample COPY -sn BAR -co target_db  target schema_map "
  ((BAR,FOO))" tablespace_map "((TS1, TS2),(TS3, TS4), SYS_ANY)"
  -u userid -p password
```

This indicates that table space TS1 is mapped to TS2, TS3 is mapped to TS4, but the remaining table spaces use a default table space selection algorithm.

Example 3: Changing the object owners after the COPY operation

You can change the owner of each new object created in the target schema after a successful COPY. The default owner of the target objects is the connect user. If this option is specified, ownership is transferred to a new owner as demonstrated:

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" tablespace_map "(SYS_ANY)" owner jrichards
  -u userid -p password
```

The new owner of the target objects is jrichards.

The **db2move** utility must be started on the target system if source and target schemas are found on different systems. For copying schemas from one database to another, this action requires a list of schema names to be copied from a source database, separated by commas, and a target database name.

To copy a schema, issue **db2move** from an operating system command prompt as follows:

```
db2move dbname COPY -co COPY-options
 -u userid -p password
```

Moving tables online by using the ADMIN_MOVE_TABLE procedure

Using the ADMIN_MOVE_TABLE procedure, you can move tables by using an online or offline move. Use an online table move instead of an offline table move if you value availability more than cost, space, move performance, and transaction overhead.

Before you begin

Ensure there is sufficient disk space to accommodate the copies of the table and index, the staging table, and the additional log entries.

About this task

You can move a table online by calling the stored procedure once or multiple times, one call for each operation performed by the procedure. Using multiple calls provides you with additional options, such as cancelling the move or controlling when the target table is taken offline to be updated.

When you call the SYSPROC.ADMIN_MOVE_TABLE procedure, a shadow copy of the source table is created. During the copy phase, changes to the source table (updates, insertions, or deletions) are captured using triggers and placed in a staging table. After the copy phase is completed, the changes captured in the staging table are replayed to the shadow copy. Following that, the stored procedure briefly takes the source table offline and assigns the source table name and index names to the shadow copy and its indexes. The shadow table is then brought online, replacing the source table. By default, the source table is dropped, but you can use the KEEP option to retain it under a different name.

Avoid performing online moves for tables without indexes, particularly unique indexes. Performing a online move for a table without a unique index might result in deadlocks and complex or expensive replay.

Procedure

To move a table online:

1. Call the ADMIN_MOVE_TABLE procedure in one of the following ways:
   - Call the ADMIN_MOVE_TABLE procedure once, specifying at least the schema name of the source table, the source table name, and an operation type of MOVE. For example, use the following syntax to move the data to an existing table within the same table space:

     ```
     CALL SYSPROC.ADMIN_MOVE_TABLE (
     'schema name',
     'source table',
     '',
     '',
     '',
     '',
     '',
     '',
     '',
     '',
     'MOVE')
     ```
   - Call the ADMIN_MOVE_TABLE procedure multiple times, once for each operation, specifying at least the schema name of the source table, the source table name, and an operation name. For example, use the following syntax to move the data to a new table within the same table space:

     ```
     CALL SYSPROC.ADMIN_MOVE_TABLE (
     'schema name',
     'source table',
     '',
     '',
     '',
     '',
     '',
     '',
     '',
     '',
     'operation name')
     ```

     where *operation name* is one of the following values: INIT, COPY, REPLAY, VERIFY, or SWAP. You must call the procedure based on this order of operations, for example, you must specify INIT as the operation name in the first call.

     Note: The VERIFY operation is costly; perform this operation only if you require it for your table move.

2. If the online move fails, rerun it:
   a. Fix the problem that caused the table move to fail.
   b. Determine the stage that was in progress when the table move failed by querying the SYSTOOLS.ADMIN_MOVE_TABLE protocol table for the status.
   c. Call the stored procedure again, specifying the applicable option:
      - If the status of the procedure is INIT, use the INIT option.
      - If the status of the procedure is COPY, use the COPY option.
      - If the status of the procedure is REPLAY, use the REPLAY or SWAP option.
      - If the status of the procedure is CLEANUP, use the CLEANUP option.

   If the status of an online table move is not COMPLETED or CLEANUP, you can cancel the move by specifying the CANCEL option for the stored procedure.

Examples

Example 1: Move the T1 table from schema SVALENTI, to the ACCOUNTING table space without taking T1 offline. Specify the DATA, INDEX, and LONG table spaces to move the table into a new table space.

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'SVALENTI',
'T1',
'ACCOUNTING',
'ACCOUNTING',
'ACCOUNTING',
'',
'',
'',
'',
'',
'MOVE')
```

Example 2: Move the T1 table from schema EBABANI to the ACCOUNTING table space without taking T1 offline, and keep a copy of the original table after the move. Use the COPY_USE_LOAD and LOAD_MSGPATH options to set the load message file path. Specify the DATA, INDEX, and LONG table spaces to move the table into a new table space. The original table will maintain a name similar to 'EBABANI'.'T1AAAAVxo'.

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'EBABANI',
'T1',
'ACCOUNTING',
'ACCOUNTING',
'ACCOUNTING',
'',
'',
'',
'',
'KEEP, COPY_USE_LOAD,LOAD_MSGPATH "/home/ebabani"',
'MOVE')
```

Example 3: Move the T1 table within the same table space. Change the C1 column within T1, which uses the deprecated datatype LONG VARCHAR to use a compatible data type.

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'SVALENTI',
'T1',
'',
'',
'',
'',
'',
'',
'C1 VARCHAR(1000), C2 INT(5), C3 CHAR(5), C4 CLOB',
'',
'MOVE')
```

Note: You cannot change the column name during this operation.

Example 4: You have the T1 table created by the following statement:

```
CREATE TABLE T1(C1 BIGINT,C2 BIGINT,C3 CHAR(20),C4 DEC(10,2),C5 TIMESTAMP,C6 BIGINT
 GENERATED ALWAYS AS (C1+c2),C7 GRAPHIC(10),C8 VARGRAPHIC(20),C9 XML
```

Move the table within the same table space and drop columns C5 and C6:

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'SVALENTI',
'T1',
'',
'',
'',
'',
'',
'',
'c1 BIGINT,c2 BIGINT ,c3 CHAR(20),c4 DEC(10,2),c7 GRAPHIC(10),c8 VARGRAPHIC(20),c9 XML',
'',
'MOVE')
```

Example 5: You have a range partitioned table with two ranges defined in tablespaces TS1 and TS2. Move the table to tablespace TS3, but leave the first range in TS1.

```
CREATE TABLE "EBABANI "."T1"  (
 "I1" INTEGER ,
 "I2" INTEGER )
  DISTRIBUTE BY HASH("I1")
 PARTITION BY RANGE("I1")
 (PART "PART0" STARTING(0) ENDING(100) IN "TS1",
 PART "PART1" STARTING(101) ENDING(MAXVALUE) IN "TS2");
```

Move the T1 table from schema EBABANI to the TS3 table space. Specify the partition definitions.

```
DB2 "CALL SYSPROC.ADMIN_MOVE_TABLE
('EBABANI',
'T1',
'TS3',
'TS3',
'TS3',
'',
'',
'(I1) (STARTING 0 ENDING 100 IN TS1 INDEX IN TS1 LONG IN TS1,
 STARTING 101 ENDING MAXVALUE IN TS3 INDEX IN TS3 LONG IN TS3)',
'',
'',
'MOVE')"
```

Mimicking databases using db2look

There are many times when it is advantageous to be able to create a database that is similar in structure to another database. For example, rather than testing out new applications or recovery plans on a production system, it makes more sense to create a test system that is similar in structure and data, and to then do the tests against it instead.

This way, the production system will not be affected by the adverse performance impact of the tests or by the accidental destruction of data by an errant application. Also, when you are investigating a problem (such as invalid results, performance issues, and so on), it might be easier to debug the problem on a test system that is identical to the production system.

You can use the **db2look** tool to extract the required DDL statements needed to reproduce the database objects of one database in another database. The tool can also generate the required SQL statements needed to replicate the statistics from the one database to the other, as well as the statements needed to replicate the database configuration, database manager configuration, and registry variables. This is important because the new database might not contain the exact same set of data as the original database but you might still want the same access plans chosen for the two systems. The **db2look** command should only be issued on databases running on DB2 Servers of Version 9.5 and higher levels.

The **db2look** tool is described in detail in the *DB2 Command Reference* but you can view the list of options by executing the tool without any parameters. A more detailed usage can be displayed using the **-h** option.

Using db2look to mimic the tables in a database

To extract the DDL for the tables in the database, use the **-e** option. For example, create a copy of the SAMPLE database called SAMPLE2 such that all of the objects in the first database are created in the new database:

```
C:\>db2 create database sample2
DB20000I The CREATE DATABASE command completed successfully.
C:\>db2look -d sample -e > sample.ddl
-- USER is:
-- Creating DDL for table(s)
-- Binding package automatically ...
-- Bind is successful
-- Binding package automatically ...
-- Bind is successful
```

Note: If you want the DDL for the user-defined spaces, database partition groups and buffer pools to be produced as well, add the**-l** flag after **-e** in the preceding command. The default database partition groups, buffer pools, and table spaces will not be extracted. This is because they already exist in every database by default. If you want to mimic these, you must alter them yourself manually.

Bring up the file `sample.ddl` in a text editor. Since you want to run the DDL in this file against the new database, you must change the CONNECT TO SAMPLE statement to CONNECT TO SAMPLE2. If you used the **-l** option, you might need to alter the path associated with the table space commands, such that they point to appropriate paths as well. While you are at it, take a look at the rest of the contents of the file. You should see CREATE TABLE, ALTER TABLE, and CREATE INDEX statements for all of the user tables in the sample database:

```
...
------------------------------------------------
-- DDL Statements for table "DB2"."ORG"
------------------------------------------------

 CREATE TABLE "DB2"."ORG"  (
    "DEPTNUMB" SMALLINT NOT NULL ,
    "DEPTNAME" VARCHAR(14) ,
    "MANAGER" SMALLINT ,
    "DIVISION" VARCHAR(10) ,
    "LOCATION" VARCHAR(13) )
   IN "USERSPACE1" ;
...
```

Once you have changed the connect statement, run the statements, as follows:

```
C:\>db2 -tvf sample.ddl > sample2.out
```

Take a look at the `sample2.out` output file -- everything should have been executed successfully. If errors have occurred, the error messages should state what the problem is. Fix those problems and run the statements again.

As you can see in the output, DDL for all of the user tables are exported. This is the default behavior but there are other options available to be more specific about the tables included. For example, to only include the STAFF and ORG tables, use the **-t** option:

```
C:\>db2look -d sample -e -t staff org > staff_org.ddl
```

To only include tables with the schema DB2, use the **-z** option:

```
C:\>db2look -d sample -e -z db2 > db2.ddl
```

Mimicking statistics for tables

If the intent of the test database is to do performance testing or to debug a performance problem, it is essential that access plans generated for both databases are identical. The optimizer generates access plans based on statistics, configuration parameters, registry variables, and environment variables. If these things are identical between the two systems then it is very likely that the access plans will be the same.

If both databases have the exact same data loaded into them and the same options of RUNSTATS is performed on both, the statistics should be identical. However, if the databases contain different data or if only a subset of data is being used in the test database then the statistics will likely be very different. In such a case, you can use **db2look** to gather the statistics from the production database and place them into the test database. This is done by creating UPDATE statements against the SYSSTAT set of updatable catalog tables as well as RUNSTATS commands against all of the tables.

The option for creating the statistic statements is **-m**. Going back to the SAMPLE/SAMPLE2 example, gather the statistics from SAMPLE and add them into SAMPLE2:

```
C:\>db2look -d sample -m > stats.dml
-- USER is:
-- Running db2look in mimic mode
```

As before, the output file must be edited such that the CONNECT TO SAMPLE statement is changed to CONNECT TO SAMPLE2. Again, take a look at the rest of the file to see what some of the RUNSTATS and UPDATE statements contain:

```
...
-- Mimic table ORG
RUNSTATS ON TABLE "DB2"."ORG" ;

UPDATE SYSSTAT.INDEXES
SET NLEAF=-1,
    NLEVELS=-1,
    FIRSTKEYCARD=-1,
    FIRST2KEYCARD=-1,
    FIRST3KEYCARD=-1,
    FIRST4KEYCARD=-1,
    FULLKEYCARD=-1,
    CLUSTERFACTOR=-1,
    CLUSTERRATIO=-1,
    SEQUENTIAL_PAGES=-1,
    PAGE_FETCH_PAIRS='',
    DENSITY=-1,
    AVERAGE_SEQUENCE_GAP=-1,
    AVERAGE_SEQUENCE_FETCH_GAP=-1,
    AVERAGE_SEQUENCE_PAGES=-1,
    AVERAGE_SEQUENCE_FETCH_PAGES=-1,
    AVERAGE_RANDOM_PAGES=-1,
    AVERAGE_RANDOM_FETCH_PAGES=-1,
    NUMRIDS=-1,
    NUMRIDS_DELETED=-1,
    NUM_EMPTY_LEAFS=-1
WHERE TABNAME = 'ORG' AND TABSCHEMA = 'DB2   ';
...
```

As with the **-e** option that extracts the DDL, the **-t** and **-z** options can be used to specify a set of tables.

Extracting configuration parameters and environment variables

The optimizer chooses plans based on statistics, configuration parameters, registry variables, and environment variables. As with the statistics, **db2look** can be used to generate the necessary configuration update and set statements. This is done using the **-f** option. For example:

```
c:\>db2look -d sample -f>config.txt
-- USER is: DB2INST1
-- Binding package automatically ...
-- Bind is successful
-- Binding package automatically ...
-- Bind is successful
```

The `config.txt` contains output similar to the following example:

```
-- This CLP file was created using DB2LOOK Version 9.1
-- Timestamp: 2/16/2006 7:15:17 PM
-- Database Name: SAMPLE
-- Database Manager Version: DB2/NT Version 9.1.0
-- Database Codepage: 1252
-- Database Collating Sequence is: UNIQUE


CONNECT TO SAMPLE;

--------------------------------------------------------
-- Database and Database Manager configuration parameters
--------------------------------------------------------

UPDATE DBM CFG USING cpuspeed 2.991513e-007;
UPDATE DBM CFG USING intra_parallel NO;
UPDATE DBM CFG USING comm_bandwidth 100.000000;
UPDATE DBM CFG USING federated NO;
```

```
...

--------------------------------
-- Environment Variables settings
--------------------------------


COMMIT WORK;

CONNECT RESET;
```

Note: Only those parameters and variables that affect DB2 compiler will be included. If a registry variable that affects the compiler is set to its default value, it will not show up under "Environment Variables settings".

Converting non-Unicode databases to Unicode

There are some cases where you might need to convert an existing non-Unicode database to a Unicode database.

Before you begin

You must have enough free disk space to export the data from the non-Unicode database. Also, if you are not reusing the existing table spaces, you need enough free disk space to create new table spaces for the data.

Procedure

To convert an existing non-Unicode database to a Unicode database:

1. Export your data using the **db2move** command:

   ```
   cd export-dir
   db2move sample export
   ```

   where *export-dir* is the directory to which you want to export your data and SAMPLE is the existing database name.
2. Generate a DDL script for your existing database using the **db2look** command:

   ```
   db2look -d sample -e -o unidb.ddl -l -x -f
   ```

   where SAMPLE is the existing database name and `unidb.ddl` is the file name for the generated DDL script. The **-l** option generates DDL for user-defined table spaces, database partition groups, and buffer pools, the **-x** option generates authorization DDL, and the **-f** option generates an update command for database configuration parameters.
3. Create the Unicode database:

   ```
   CREATE DATABASE UNIDB COLLATE USING SYSTEM_codepage_territory
   ```

   where *UNIDB* is the name of the Unicode database and `SYSTEM_`*`codepage_territory`* is a language-aware collation based on the weight table used for collating your non-Unicode data. This ensures that the data in the new Unicode database is sorted in the same order.
4. Edit the `unidb.ddl` script:
   a. Change all occurrences of the database name to the new Unicode database name:

      ```
      CONNECT TO UNIDB
      ```

b. Increase the column lengths for character columns in your tables. When characters are converted to Unicode, there might be an expansion in the number of bytes. It is recommended that you increase the length of the character columns to compensate for this expansion.

c. To keep the existing database, you must also change the file name specification for table spaces in the `unidb.ddl` file. Otherwise, you can drop the existing database and use the same table space files:

```
DROP DATABASE SAMPLE
```

5. Recreate your database structure by running the DDL script that you edited:

```
db2 -tvf unidb.ddl
```

6. Import your data into the new Unicode database using the **db2move** command:

```
cd export-dir
db2move unidb import
```

where *export-dir* is the directory where you exported your data and UNIDB is the Unicode database name.

Creating database duplicates

Creating production database duplicates in a test environment allows you to test upgrading your databases before you upgrade them in your production environment.

Before you begin

Ensure that you have SYSCTRL or SYSADM authority.

About this task

This procedure uses DDL scripts to create database duplicates. If you have enough resources, you can also create database duplicates by restoring a database backup to create a new database. See “Restoring to a new database” in *Data Recovery and High Availability Guide and Reference* for details.

Procedure

To create a database duplicate for testing database upgrade:

1. Log on as the instance owner on the production database server and use the **db2look** command to generate DDL scripts with all the existing objects in your databases. The following command shows how to generate the `sample.ddl` script for the SAMPLE database:

```
db2look -d sample -a -e -m -l -x -f -o sample.ddl
```

Edit the generated DDL scripts and change:

- The database name in the CONNECT statements
- The path of the user table space containers or data and reduce the sizes to a minimum size since to re-create a database with no data or just a data subset

You can use your own DDL scripts to create test databases in the test instance instead of generating DDL scripts.

2. Log on as the instance owner in the test database server and create your database duplicates. The following example shows how to create a database duplicate of the SAMPLE database using the `sample.ddl` script:

```
db2 CREATE DATABASE NSAMPLE
db2 -tvsf sample.ddl
db2 UPDATE DBM CONFIGURATION USING diaglevel 4
```

All significant upgrade events are logged in the **db2diag** log files when the **diaglevel** database manager configuration parameter is set to 3 (default value) or higher. A value of 4 captures additional information that can be helpful in problem determination.

3. Adjust the size of the system catalog table space, temporary table space, and log space in your test databases if required.
4. Export data subsets of your production databases and import these data subsets into your test databases. For details, see “Exporting Data” and “Importing Data” in *Data Movement Utilities Guide and Reference*. You only need a data subset if you are going to test your applications in your testing environment.
5. Verify that your database duplicates were created successfully by connecting to the them and issue a small query.

Chapter 39. Data organization

Over time, data in your tables can become fragmented, increasing the size of tables and indexes as records become distributed over more and more data pages. This can increase the number of pages that need to be read during query execution. Reorganization of tables and indexes compacts your data, reclaiming wasted space and improving data access.

Procedure

The steps to perform an index or table reorganization are as follows:
1. Determine whether you need to reorganize any tables or indexes.
2. Choose a reorganization method.
3. Perform the reorganization of identified objects.
4. Optional: Monitor the progress of reorganization.
5. Determine whether or not the reorganization was successful. For offline table reorganization and any index reorganization, the operation is synchronous, and the outcome is apparent upon completion of the operation. For online table reorganization, the operation is asynchronous, and details are available from the history file.
6. Collect statistics on reorganized objects.
7. Rebind applications that access reorganized objects.

Table reorganization

After many changes to table data, logically sequential data might reside on nonsequential data pages, so that the database manager might need to perform additional read operations to access data. Also, if many rows have been deleted, additional read operations are also required. In this case, you might consider reorganizing the table to match the index and to reclaim space.

You can also reorganize the system catalog tables.

Because reorganizing a table usually takes more time than updating statistics, you could execute the **RUNSTATS** command to refresh the current statistics for your data, and then rebind your applications. If refreshed statistics do not improve performance, reorganization might help.

The following factors can indicate a need for table reorganization:
- There has been a high volume of insert, update, and delete activity against tables that are accessed by queries.
- There have been significant changes in the performance of queries that use an index with a high cluster ratio.
- Executing the **RUNSTATS** command to refresh table statistics does not improve performance.
- Output from the **REORGCHK** command indicates a need for table reorganization.

Note: With DB2 V9.7 Fix Pack 1 and later releases, higher data availability for a data partitioned table with only partitioned indexes (except system-generated XML path indexes) is achieved by reorganizing data for a specific data partition.

Partition-level reorganization performs a table reorganization on a specified data partition while the remaining data partitions of the table remain accessible. The output from the **REORGCHK** command for a partitioned table contains statistics and recommendations for performing partition-level reorganizations.

REORG TABLE commands and **REORG INDEXES ALL** commands can be issued on a data partitioned table to concurrently reorganize different data partitions or partitioned indexes on a partition. When concurrently reorganizing data partitions or the partitioned indexes on a partition, users can access the unaffected partitions but cannot access the affected partitions. All the following criteria must be met to issue REORG commands that operate concurrently on the same table:

- Each REORG command must specify a different partition with the **ON DATA PARTITION** clause.
- Each REORG command must use the ALLOW NO ACCESS mode to restrict access to the data partitions.
- The partitioned table must have only partitioned indexes if issuing **REORG TABLE** commands. No nonpartitioned indexes (except system-generated XML path indexes) can be defined on the table.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for reorganizing tables. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Choosing a table reorganization method

There are two approaches to table reorganization: *classic reorganization* (offline) and *inplace reorganization* (online).

Offline reorganization is the default behavior. To specify an online reorganization operation, use the INPLACE option on the **REORG TABLE** command.

An alternative approach to inplace reorganization, using online table move stored procedures, is also available. See “Moving tables online by using the ADMIN_MOVE_TABLE procedure”.

Each approach has its advantages and drawbacks, which are summarized in the following sections. When choosing a reorganization method, consider which approach offers advantages that align with your priorities. For example, if recoverability in case of failure is more important than performance, online reorganization might be preferable.

Advantages of offline reorganization

This approach offers:

- The fastest table reorganization operations, especially if large object (LOB) or long field data is not included
- Perfectly clustered tables and indexes upon completion
- Indexes that are automatically rebuilt after a table has been reorganized; there is no separate step for rebuilding indexes
- The use of a temporary table space for building a shadow copy; this reduces the space requirements for the table space that contains the target table or index
- The use of an index other than the clustering index to re-cluster the data

Disadvantages of offline reorganization

This approach is characterized by:

- Limited table access; read access only during the sort and build phase of a reorg operation
- A large space requirement for the shadow copy of the table that is being reorganized
- Less control over the reorg process; an offline reorg operation cannot be paused and restarted

Advantages of online reorganization

This approach offers:

- Full table access, except during the truncation phase of a reorg operation
- More control over the reorg process, which runs asynchronously in the background, and which can be paused, resumed, or stopped; for example, you can pause an in-progress reorg operation if a large number of update or delete operations are running against the table
- A recoverable process in the event of a failure
- A reduced requirement for working storage, because a table is processed incrementally
- Immediate benefits of reorganization, even before a reorg operation completes

Disadvantages of online reorganization

This approach is characterized by:

- Imperfect data or index clustering, depending on the type of transactions that access the table during a reorg operation
- Poorer performance than an offline reorg operation
- Potentially high logging requirements, depending on the number of rows being moved, the number of indexes that are defined on the table, and the size of those indexes
- A potential need for subsequent index reorganization, because indexes are maintained, not rebuilt
- Incomplete space reclamation, because online reorganization cannot move internal records.

Table 117. Comparison of online and offline reorganization

Characteristic	Offline reorganization	Online reorganization
Performance	Fast	Slow
Clustering factor of data at completion	Good	Not perfectly clustered
Concurrency (access to the table)	Ranges from no access to read-only	Ranges from read-only to full access
Data storage space requirement	Significant	Not significant
Logging storage space requirement	Not significant	Could be significant
User control (ability to pause, restart process)	Less control	More control

Table 117. Comparison of online and offline reorganization (continued)

Characteristic	Offline reorganization	Online reorganization
Recoverability	Not recoverable	Recoverable
Index rebuilding	Done	Not done
Supported for all types of tables	Yes	No
Ability to specify an index other than the clustering index	Yes	No
Use of a temporary table space	Yes	No

Table 118. Table types that are supported for online and offline reorganization

Table type	Offline reorganization supported	Online reorganization supported
Multidimensional clustering tables (MDC)	Yes[1]	No
Insert time clustering tables (ITC)	Yes[1]	No
Range-clustered tables (RCT)	No[2]	No
Append mode tables	Yes	No[3]
Tables with long field or large object (LOB) data	Yes[4]	Yes[5]
System catalog tables: • SYSIBM.SYSCODEPROPERTIES • SYSIBM.SYSDATATYPES • SYSIBM.SYSNODEGROUPS • SYSIBM.SYSROUTINES • SYSIBM.SYSSEQUENCES • SYSIBM.SYSTABLES • SYSIBM.SYSVARIABLES	Yes	No

Notes:

1. Because clustering is automatically maintained through MDC block indexes, reorganization of an MDC table involves space reclamation only. No indexes can be specified. Similarly, for ITC tables, you cannot specify a reorganization using a clustering index.
2. The range area of an RCT always remains clustered.
3. Online reorganization can be performed after append mode is disabled.
4. Reorganizing long field or large object (LOB) data can take a significant amount of time, and does not improve query performance; it should only be done for space reclamation.
5. Online table reorganization does not reorganize the LONG/LOB data, but reorganizes the other columns.

Monitoring the progress of table reorganization

Information about the progress of a current table reorg operation is written to the history file. The history file contains a record for each reorganization event. To view this file, execute the **LIST HISTORY** command against the database that contains the table being reorganized.

You can also use table snapshots to monitor the progress of table reorg operations. Table reorganization monitoring data is recorded, regardless of the setting for the database system monitor table switch.

If an error occurs, an SQLCA message is written to the history file. In the case of an inplace table reorg operation, the status is recorded as PAUSED.

Classic (offline) table reorganization

Classic table reorganization uses a shadow copy approach, building a full copy of the table that is being reorganized.

There are four phases in a classic or offline table reorganization operation:

1. SORT - During this phase, if an index was specified on the **REORG TABLE** command, or a clustering index was defined on the table, the rows of the table are first sorted according to that index. If the INDEXSCAN option is specified, an index scan is used to sort the table; otherwise, a table scan sort is used. This phase applies only to a clustering table reorg operation. Space reclaiming reorg operations begin at the build phase.
2. BUILD - During this phase, a reorganized copy of the entire table is built, either in its table space or in a temporary table space that was specified on the **REORG TABLE** command.
3. REPLACE - During this phase, the original table object is replaced by a copy from the temporary table space, or a pointer is created to the newly built object within the table space of the table that is being reorganized.
4. RECREATE ALL INDEXES - During this phase, all indexes that were defined on the table are recreated.

You can monitor the progress of the table reorg operation and identify the current phase using the snapshot monitor or snapshot administrative views.

The locking conditions are more restrictive in offline mode than in online mode. Read access to the table is available while the copy is being built. However, exclusive access to the table is required when the original table is being replaced by the reorganized copy, or when indexes are being rebuilt.

An IX table space lock is required during the entire table reorg process. During the build phase, a U lock is acquired and held on the table. A U lock allows the lock owner to update the data in the table. Although no other application can update the data, read access is permitted. The U lock is upgraded to a Z lock after the replace phase starts. During this phase, no other applications can access the data. This lock is held until the table reorg operation completes.

A number of files are created by the offline reorganization process. These files are stored in your database directory. Their names are prefixed with the table space and object IDs; for example, `0030002.ROR` is the state file for a table reorg operation whose table space ID is 3 and table ID is 2.

The following list shows the temporary files that are created in a system managed space (SMS) table space during an offline table reorg operation:

- .DTR - Data shadow copy file
- .LFR - Long field file
- .LAR - Long field allocation file
- .RLB - LOB data file
- .RBA - LOB allocation file
- .BMR - Block object file for multidimensional clustering (MDC) and insert time clustering (ITC) tables

The following temporary file is created during an index reorg operation:

- .IN1 - Shadow copy file

The following list shows the temporary files that are created in the system temporary table space during the sort phase:

- .TDA - Data file
- .TIX - Index file
- .TLF - Long field file
- .TLA - Long field allocation file
- .TLB - LOB file
- .TBA - LOB allocation file
- .TBM - Block object file

The files that are associated with the reorganization process should not be manually removed from your system.

Reorganizing tables offline

Reorganizing tables offline is the fastest way to defragment your tables. Reorganization reduces the amount of space that is required for a table and improves data access and query performance.

Before you begin

You must have SYSADM, SYSCTRL, SYSMAINT, DBADM, or SQLADM authority, or CONTROL privilege on the table that is to be reorganized. You must also have a database connection to reorganize a table.

About this task

After you have identified the tables that require reorganization, you can run the reorg utility against those tables and, optionally, against any indexes that are defined on those tables.

Procedure

1. To reorganize a table using the **REORG TABLE** command, simply specify the name of the table. For example:

   ```
   reorg table employee
   ```

   You can reorganize a table using a specific temporary table space. For example:

   ```
   reorg table employee use mytemp
   ```

You can reorganize a table and have the rows reordered according to a specific index. For example:

```
reorg table employee index myindex
```

2. To reorganize a table using an SQL CALL statement, specify the **REORG TABLE** command with the ADMIN_CMD procedure. For example:

   ```
   call sysproc.admin_cmd ('reorg table employee')
   ```

3. To reorganize a table using the administrative application programming interface, call the db2Reorg API.

What to do next

After reorganizing a table, collect statistics on that table so that the optimizer has the most accurate data for evaluating query access plans.

Inplace (online) table reorganization

Inplace table reorganization enables you to reorganize a table while you have full access to its data. The cost of this uninterrupted access to the data is a slower table reorg operation.

Restriction: Inplace (online) reorganization is not supported in DB2 pureScale environments. Any attempts to perform an inplace reorganization in DB2 pureScale environments will fail with SQL1419N.

During an inplace or online table reorg operation, portions of a table are reorganized sequentially. Data is not copied to a temporary table space; instead, rows are moved within the existing table object to reestablish clustering, reclaim free space, and eliminate overflow rows.

There are four main phases in an online table reorg operation:

1. SELECT *n* pages

   During this phase, the database manager selects a range of *n* pages, where *n* is the size of an extent with a minimum of 32 sequential pages for reorg processing.

2. Vacate the range

   The reorg utility moves all rows within this range to free pages in the table. Each row that is moved leaves behind a reorg table pointer (RP) record that contains the record ID (RID) of the row's new location. The row is placed on a free page in the table as a reorg table overflow (RO) record that contains the data. After the utility has finished moving a set of rows, it waits until all applications that are accessing data in the table are finished. These “old scanners” use old RIDs when accessing the table data. Any table access that starts during this waiting period (a “new scanner”) uses new RIDs to access the data. After all of the old scanners have completed, the reorg utility cleans up the moved rows, deleting RP records and converting RO records into regular records.

3. Fill the range

   After all rows in a specific range have been vacated, they are written back in a reorganized format, sorted according to any indexes that were used, and obeying any PCTFREE restrictions that were defined. When all of the pages in the range have been rewritten, the next *n* sequential pages in the table are selected, and the process is repeated.

4. Truncate the table

By default, when all pages in the table have been reorganized, the table is truncated to reclaim space. If the NOTRUNCATE option has been specified, the reorganized table is not truncated.

Files created during an online table reorg operation

During an online table reorg operation, an `.OLR` state file is created for each database partition. This binary file has a name whose format is `xxxxyyyy.OLR`, where *xxxx* is the table space ID and *yyyy* is the object ID in hexadecimal format. This file contains the following information that is required to resume an online reorg operation from the paused state:

- The type of reorg operation
- The life log sequence number (LSN) of the table being reorganized
- The next range to be vacated
- Whether the reorg operation is clustering the data or just reclaiming space
- The ID of the index that is being used to cluster the data

A checksum is performed on the `.OLR` file. If the file becomes corrupted, causing checksum errors, or if the table LSN does not match the life LSN, a new reorg operation is initiated, and a new state file is created.

If the `.OLR` state file is deleted, the reorg process cannot resume, SQL2219N is returned, and a new reorg operation must be initiated.

The files that are associated with the reorganization process should not be manually removed from your system.

Locking and concurrency considerations for online table reorganization

One of the most important aspects of online table reorganization-because it is so crucial to application concurrency-is how locking is controlled.

An online table reorg operation can hold the following locks:

- To ensure write access to table spaces, an IX lock is acquired on the table spaces that are affected by the reorg operation.
- A table lock is acquired and held during the entire reorg operation. The level of locking is dependent on the access mode that is in effect during reorganization:
  - If ALLOW WRITE ACCESS was specified, an IS table lock is acquired.
  - If ALLOW READ ACCESS was specified, an S table lock is acquired.
- An S lock on the table is requested during the truncation phase. Until the S lock is acquired, rows can be inserted by concurrent transactions. These inserted rows might not be seen by the reorg utility, and could prevent the table from being truncated. After the S table lock is acquired, rows that prevent the table from being truncated are moved to compact the table. After the table is compacted, it is truncated, but only after all transactions that are accessing the table at the time the truncation point is determined have completed.
- A row lock might be acquired, depending on the type of table lock:
  - If an S lock is held on the table, there is no need for individual row-level S locks, and further locking is unnecessary.
  - If an IS lock is held on the table, an NS row lock is acquired before the row is moved, and then released after the move is complete.
- Certain internal locks might also be acquired during an online table reorg operation.

Locking has an impact on the performance of both online table reorg operations and concurrent user applications. You can use lock snapshot data to help you to understand the locking activity that occurs during online table reorganizations.

Reorganizing tables online

An online or inplace table reorganization allows users to access a table while it is being reorganized.

Before you begin

You must have SYSADM, SYSCTRL, SYSMAINT, DBADM, or SQLADM authority, or CONTROL privilege on the table that is to be reorganized. You must also have a database connection to reorganize a table.

About this task

After you have identified the tables that require reorganization, you can run the reorg utility against those tables and, optionally, against any indexes that are defined on those tables.

Procedure

- To reorganize a table online using the **REORG TABLE** command, specify the name of the table and the **INPLACE** parameter. For example:

  ```
  reorg table employee inplace
  ```
- To reorganize a table online using an SQL CALL statement, specify the **REORG TABLE** command with the ADMIN_CMD procedure. For example:

  ```
  call sysproc.admin_cmd ('reorg table employee inplace')
  ```
- To reorganize a table online using the administrative application programming interface, call the db2Reorg API.

What to do next

After reorganizing a table, collect statistics on that table so that the optimizer has the most accurate data for evaluating query access plans.

Pausing and restarting an online table reorganization

An online table reorganization that is in progress can be paused and restarted by the user.

Before you begin

You must have SYSADM, SYSCTRL, SYSMAINT, DBADM, or SQLADM authority, or CONTROL privilege on the table whose online reorganization is to be paused or restarted. You must also have a database connection to pause or restart an online table reorganization.

Procedure

1. To pause an online table reorganization using the **REORG TABLE** command, specify the name of the table, the **INPLACE** parameter, and the **PAUSE** parameter. For example:

   ```
   reorg table employee inplace pause
   ```
2. To restart a paused online table reorganization, specify the **RESUME** parameter. For example:

   ```
   reorg table employee inplace resume
   ```

When an online table reorg operation is paused, you cannot begin a new reorganization of that table. You must either resume or stop the paused operation before beginning a new reorganization process.

Following a **RESUME** request, the reorganization process respects whatever truncation option is specified on the current **RESUME** request. For example, if the **NOTRUNCATE** parameter is not specified on the current **RESUME** request, a **NOTRUNCATE** parameter specified on the original **REORG TABLE** command, or with any previous **RESUME** requests, is ignored.

A table reorg operation cannot resume after a restore and rollforward operation.

Monitoring a table reorganization

You can use the **GET SNAPSHOT** command, the SNAPTAB_REORG administrative view, or the SNAP_GET_TAB_REORG table function to obtain information about the status of your table reorganization operations.

Procedure

- To access information about reorganization operations using SQL, use the SNAPTAB_REORG administrative view. For example, the following query returns details about table reorganization operations on all database partitions for the currently connected database. If no tables have been reorganized, no rows are returned.

```
select
    substr(tabname, 1, 15) as tab_name,
    substr(tabschema, 1, 15) as tab_schema,
    reorg_phase,
    substr(reorg_type, 1, 20) as reorg_type,
    reorg_status,
    reorg_completion,
    dbpartitionnum
  from sysibmadm.snaptab_reorg
  order by dbpartitionnum
```

- To access information about reorganization operations using the snapshot monitor, use the **GET SNAPSHOT FOR TABLES** command and examine the values of the table reorganization monitor elements.

Results

Because offline table reorg operations are synchronous, errors are returned to the caller of the utility (an application or the command line processor). And because online table reorg operations are asynchronous, error messages in this case are not returned to the CLP. To view SQL error messages that are returned during an online table reorg operation, use the **LIST HISTORY REORG** command.

An online table reorg operation runs in the background as the db2Reorg process. This process continues running even if the calling application terminates its database connection.

Index reorganization

As tables are updated, index performance can degrade.

The degradation can occur in the following ways:

- Leaf pages become fragmented. When leaf pages are fragmented, I/O costs increase because more leaf pages must be read to fetch table pages.

- The physical index page order no longer matches the sequence of keys on those pages, resulting in low density indexes. When leaf pages have a low density, sequential prefetching is inefficient and the number of I/O waits increases. However, if smart index prefetching is enabled, the query optimizer switches to readahead prefetching if low density indexes exist. This helps reduce the negative impact that low density indexes have on performance.
- The index develops too many levels. In this case, the index should be reorganized.

Index reorganization requires:
- SYSADM, SYSMAINT, SYSCTRL, DBADM, or SQLADM authority, or CONTROL privilege on the table and its indexes
- When the REBUILD option with the ALLOW READ or WRITE ACCESS options are chosen, an amount of free space in the table space where the indexes are stored is required. This space must be equal to the current size of the indexes. Consider placing indexes in a large table space when you issue the **CREATE TABLE** statement.
- Additional log space. The index reorg utility logs its activities.

If you specify the MINPCTUSED option on the CREATE INDEX statement, the database server automatically merges index leaf pages if a key is deleted and the free space becomes less than the specified value. This process is called *online index defragmentation*.

To restore index clustering, free up space, and reduce leaf levels, you can use one of the following methods:
- Drop and recreate the index.
- Use the **REORG TABLE** command with options that allow you to reorganize the table and rebuild its indexes offline.
- Use the **REORG INDEXES** command with the REBUILD option to reorganize indexes online or offline. You might choose online reorganization in a production environment, because it allows users to read from or write to the table while its indexes are being rebuilt.

If your primary objective is to free up space, consider using the CLEANUP and RECLAIM EXTENTS options of the **REORG** command. See the related links for more details.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for reorganizing indexes. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

With DB2 V9.7 Fix Pack 1 and later releases, using the **REORG INDEXES ALL** command on a data partitioned table and specifying a partition with the ON DATA PARTITION clause reorganizes the partitioned indexes for single data partition. During index reorganization, the unaffected partitions remain read and write accessible access is restricted only to the affected partition.

REORG TABLE commands and **REORG INDEXES ALL** commands can be issued on a data partitioned table to concurrently reorganize different data partitions or partitioned indexes on a partition. When concurrently reorganizing data partitions or the partitioned indexes on a partition, users can access the unaffected partitions. All the following criteria must be met to issue REORG commands that operate concurrently on the same table:

- Each REORG command must specify a different partition with the **ON DATA PARTITION** clause.
- Each REORG command must use the ALLOW NO ACCESS mode to restrict access to the data partitions.
- The partitioned table must have only partitioned indexes if issuing **REORG TABLE** commands. No nonpartitioned indexes (except system-generated XML path indexes) can be defined on the table.

Note: The output from the **REORGCHK** command contains statistics and recommendations for reorganizing indexes. For a partitioned table, the output contains statistics and recommendations for reorganizing partitioned and nonpartitioned indexes. Alternatively, if the objective is to reclaim space, the **RECLAIMABLE_SPACE** output of the **ADMIN_GET_INDEX_INFO** function shows how much space is reclaimable. Use the **REORG INDEXES** command with the RECLAIM EXTENTS option to free this reclaimable space.

Online index reorganization

When you use the **REORG INDEXES** command with the ALLOW READ/WRITE ACCESS and REBUILD options, a shadow copy of the index object is built while the original index object remains available as read or write access to the table continues. If write access is allowed, then during reorganization, any changes to the underlying table that would affect the indexes are logged. The reorg operation processes these logged changes while rebuilding the indexes.

Changes to the underlying table that would affect the indexes are also written to an internal memory buffer, if such space is available for use. The internal buffer is a designated memory area that is allocated on demand from the utility heap. The use of a memory buffer enables the index reorg utility to process the changes by reading directly from memory first, and then reading through the logs, if necessary, but at a much later time. The allocated memory is freed after the reorg operation completes.

Extra storage space is required in the index tablespace to hold the shadow copy of the index. Once the shadow copy of the index is built and all logs affecting the shadow copy have been processed, then a super-exclusive lock is taken on the table and the original index is discarded. The space that was occupied by the original copy of the index object is free to be reused by any object in the same tablespace, however it is not automatically returned to the filesystem.

Online index reorganization in ALLOW WRITE ACCESS mode (with or without the CLEANUP option) is not supported for spatial indexes or multidimensional clustering (MDC) and insert time clustering (ITC) tables.

Restriction: Online reorganization with the REBUILD option is not supported in DB2 pureScale environments. Any attempts to perform an online reorganization with the REBUILD option in DB2 pureScale environments will fail with SQL1419N.

Locking and concurrency considerations for online index reorganization

In this topic, online index reorganization applies to an index reorganization that is run with the **ALLOW READ ACCESS** or **ALLOW WRITE ACCESS** parameters.

These options allow you to access the table while its indexes are being reorganized. During online index reorganization with the **REBUILD** option, new indexes are built as additional copies while the original indexes remain intact. Concurrent transactions use the original indexes while the new indexes are created. At the end of the reorganization operation, the original indexes are replaced by the new indexes. Transactions that are committed in the meantime are reflected in the new indexes after the replacement of the original indexes. If the reorganization operation fails and the transaction is rolled back, the original indexes remain intact. During online index reorganization with the **CLEANUP** and **RECLAIM EXTENTS** options, space is reclaimed and made available for use for all objects in the table space.

An online index reorganization operation can hold the following locks:

- To ensure access to table spaces, an IX-lock is acquired on the table spaces affected by the reorganization operation. This includes table spaces that hold the table, as well as partition, and index objects.
- To prevent the affected table from being altered during reorganization, an X alter table lock is acquired.
- A table lock is acquired and held throughout the reorganization operation. The type of lock depends on the table type, access mode, and reorganization option:
  - For nonpartitioned tables:
    - If **ALLOW READ ACCESS** is specified, a U-lock is acquired on the table.
    - If **ALLOW WRITE ACCESS** is specified, an IN-lock is acquired on the table.
    - If **CLEANUP** is specified, an S-lock is acquired on the table for READ access, and IX-lock for WRITE access.
  - For partitioned tables, reorganization with **ALLOW READ ACCESS** or **ALLOW WRITE ACCESS** is supported at partition level only:
    - If **ALLOW READ ACCESS** is specified, a U-lock is acquired on the partition.
    - If **ALLOW WRITE ACCESS** is specified, an IS-lock is acquired on the partition.
    - If **CLEANUP** is specified, an S-lock is acquired on the partition for READ access, and an IX-lock for WRITE access.
    - An IS-lock is acquired on the table regardless of which access mode or option is specified.
- An exclusive Z-lock on the table or partition is requested at the end of index reorganization. If a partitioned table contains nonpartitioned indexes, then the Z-lock is acquired on the table as well as the partition. This lock suspends table and partition access to allow for the replacement of the original indexes by the new indexes. This lock is held until transactions that are committed during reorganization are reflected in the new indexes.
- The IS table lock and NS row lock are acquired on the system catalog table SYSIBM.SYSTABLES.
- For a partition level reorganization, IS table lock and NS row lock are also acquired on the system catalog table SYSIBM.SYSDATAPARTITIONS.
- Certain internal locks might also be acquired during an online index reorganization operation.
- Online index reorganization might have impact on concurrency if the reorganization operation fails. For example, the reorganization might fail due to

insufficient memory, lack of disk space, or a lock timeout. The reorganization transaction performs certain updates before ending. To perform updates, reorganization must wait on existing transaction to be committed. This delay might block other transactions in the process. Starting in DB2 Version 9.7 Fix Pack 1, reorganization requests a special drain lock on the index object. Reorganization operations wait for existing transactions to finish; however, new requests to access the index object are allowed.

Monitoring an index reorganization operation

About this task

You can use the **db2pd** command to monitor the progress of reorganization operations on a database.

Procedure

Issue the **db2pd** command with the **-reorgs index** parameter:

```
db2pd -reorgs index
```

Results

The following is an example of output obtained using the **db2pd** command with the**-reorgs index** parameter, which reports the index reorganization progress for a range-partitioned table with two partitions.

Note: The first output reports the Index Reorg Stats of the non-partitioned indexes. The following outputs report the Index Reorg Stats of the partitioned indexes on each partition; the index reorganization statistics of only one partition is reported in each output.

```
Index Reorg Stats:
Retrieval Time: 02/08/2010 23:04:21
TbspaceID: -6       TableID: -32768
Schema: TEST1    TableName: BIGRPT
Access: Allow none
Status: Completed
Start Time: 02/08/2010 23:03:55   End Time: 02/08/2010 23:04:04
Total Duration: 00:00:08
Prev Index Duration:  -
Cur Index Start: -
Cur Index: 0            Max Index: 2            Index ID: 0
Cur Phase: 0          ( -     )   Max Phase: 0
Cur Count: 0                      Max Count: 0
Total Row Count: 750000
Retrieval Time: 02/08/2010 23:04:21
TbspaceID: 2        TableID: 5
Schema: TEST1    TableName: BIGRPT
PartitionID: 0      MaxPartition: 2
Access: Allow none
Status: Completed
Start Time: 02/08/2010 23:04:04   End Time: 02/08/2010 23:04:08
Total Duration: 00:00:04
Prev Index Duration:  -
Cur Index Start: -
Cur Index: 0            Max Index: 2            Index ID: 0
Cur Phase: 0          ( -     )   Max Phase: 0
Cur Count: 0                      Max Count: 0
Total Row Count: 375000
```

```
Retrieval Time: 02/08/2010 23:04:21
TbspaceID: 2        TableID: 6
Schema: TEST1    TableName: BIGRPT
PartitionID: 1      MaxPartition: 2
Access: Allow none
Status: Completed
Start Time: 02/08/2010 23:04:08   End Time: 02/08/2010 23:04:12
Total Duration: 00:00:04
Prev Index Duration:  -
Cur Index Start: -
Cur Index: 0            Max Index: 2            Index ID: 0
Cur Phase: 0          ( -     )   Max Phase: 0
Cur Count: 0                      Max Count: 0
Total Row Count: 375000
```

Determining when to reorganize tables and indexes

After many changes to table data, logically sequential data might be on nonsequential physical data pages, especially if many update operations created overflow records. When the data is organized in this way, the database manager must perform additional read operations to access required data. Additional read operations are also required if many rows are deleted.

About this task

Table reorganization defragments data, eliminating wasted space. It also reorders the rows to incorporate overflow records, improving data access and, ultimately, query performance. You can specify that the data can be reordered according to a particular index, so that queries can access the data with a minimal number of read operations.

Many changes to table data can cause index performance to degrade. Index leaf pages can become fragmented or badly clustered, and the index could develop more levels than necessary for optimal performance. All of these issues cause more I/Os and can degrade performance.

Any one of the following factors indicate that you might reorganize a table or index:

- A high volume of insert, update, and delete activity against a table since the table was last reorganized
- Significant changes in the performance of queries that use an index with a high cluster ratio
- Executing the **RUNSTATS** command to refresh statistical information does not improve performance
- Output from the **REORGCHK** command suggests that performance can be improved by reorganizing a table or its indexes

  In some cases, the reorgchk utility might recommend table reorganization, even after a table reorg operation is performed. You should analyze reorgchk utility recommendations and assess the potential benefits against the costs of performing a reorganization.
- If reclaiming space is your primary concern, the **REORG** command with the CLEANUP and RECLAIM EXTENTS options can be used.

  The **RECLAIMABLE_SPACE** output of the **ADMIN_GET_INDEX_INFO** and **ADMIN_GET_TAB_INFO** functions show how much space, in kilobytes, is available for reclaim. If you issue the **REORG** command with the CLEANUP option before running the **ADMIN_GET_INDEX_INFO** and **ADMIN_GET_TAB_INFO** functions, the output of the functions shows the maximum space available for reclamation. Use this information to determine when a **REORG** with RECLAIM EXTENTS would help reduce the size of your tables and indexes.

The **REORGCHK** command returns statistical information about data organization and can advise you about whether particular tables or indexes need to be reorganized. When space reclaim is your only concern, the **RECLAIMABLE_SPACE** output of the **ADMIN_GET_INDEX_INFO** and **ADMIN_GET_TAB_INFO** functions outline how much space is available for reclaim. However, running specific queries against the SYSSTAT views at regular intervals or at specific times can build a history that helps you identify trends that have potentially significant performance implications.

To determine whether there is a need to reorganize your tables or indexes, query the SYSSTAT views and monitor the following statistics:

- Overflow of rows

  Query the OVERFLOW column in the SYSSTAT.TABLES view to monitor the overflow value. The value represents the number of rows that do not fit on their original pages. Row data can overflow when variable length columns cause the record length to expand to the point that a row no longer fits into its assigned location on the data page. Length changes can also occur if a column is added to the table. In this case, a pointer is kept at the original location in the row and the actual value is stored in another location that is indicated by the pointer. This can impact performance because the database manager must follow the pointer to find the contents of the column. This two-step process increases the processing time and might also increase the number of I/Os that are required. Reorganizing the table data will eliminate any row overflows.
- Fetch statistics

  Query the following columns in the SYSSTAT.INDEXES catalog view to determine the effectiveness of the prefetchers when the table is accessed in index order. These statistics characterize the average performance of the prefetchers against the underlying table.
  - The AVERAGE_SEQUENCE_FETCH_PAGES column stores the average number of pages that can be accessed in sequence. Pages that can be accessed in sequence are eligible for prefetching. A small number indicates that the prefetchers are not as effective as they could be, because they cannot read in the full number of pages that is specified by the PREFETCHSIZE value for the table space. A large number indicates that the prefetchers are performing effectively. For a clustered index and table, this number should approach the value of NPAGES, the number of pages that contain rows.
  - The AVERAGE_RANDOM_FETCH_PAGES column stores the average number of random table pages that are fetched between sequential page accesses when fetching table rows using an index. The prefetchers ignore small numbers of random pages when most pages are in sequence, and continue to prefetch to the configured prefetch size. As the table becomes more disorganized, the number of random fetch pages increases. Disorganization is usually caused by insertions that occur out of sequence, either at the end of the table or in overflow pages, and query performance is impacted when an index is used to access a range of values.
  - The AVERAGE_SEQUENCE_FETCH_GAP column stores the average gap between table page sequences when fetching table rows using an index. Detected through a scan of index leaf pages, each gap represents the average number of table pages that must be randomly fetched between sequences of table pages. This occurs when many pages are accessed randomly, which

interrupts the prefetchers. A large number indicates that the table is disorganized or poorly clustered to the index.

- Number of index leaf pages containing record identifiers (RIDs) that are marked deleted but not yet removed

  RIDs are not usually physically deleted when they are marked deleted. This means that useful space might be occupied by these logically deleted RIDs. To retrieve the number of leaf pages on which every RID is marked deleted, query the NUM_EMPTY_LEAFS column of the SYSSTAT.INDEXES view. For leaf pages on which not all RIDs are marked deleted, the total number of logically deleted RIDs is stored in the NUMRIDS_DELETED column.

  Use this information to estimate how much space might be reclaimed by invoking the **REORG INDEXES** command with the CLEANUP ALL option. To reclaim only the space in pages on which all RIDs are marked deleted, invoke the **REORG INDEXES** command with the CLEANUP PAGES option.

- Cluster-ratio and cluster-factor statistics for indexes

  In general, only one of the indexes for a table can have a high degree of clustering. A cluster-ratio statistic is stored in the CLUSTERRATIO column of the SYSCAT.INDEXES catalog view. This value, between 0 and 100, represents the degree of data clustering in the index. If you collect detailed index statistics, a finer cluster-factor statistic between 0 and 1 is stored in the CLUSTERFACTOR column instead, and the value of CLUSTERRATIO is -1. Only one of these two clustering statistics can be recorded in the SYSCAT.INDEXES catalog view. To compare CLUSTERFACTOR values with CLUSTERRATIO values, multiply the CLUSTERFACTOR value by 100 to obtain a percentage value.

  Index scans that are not index-only access might perform better with higher density of indexes. A low density leads to more I/O for this type of scan, because a data page is less likely to remain in the buffer pool until it is accessed again. Increasing the buffer size might improve the performance of low density indexes. Also, if smart index prefetching is enabled, it can also improve the performance of low density indexes which reduces the need to perform the **REORG** command on indexes. Smart index prefetching achieves this by switching from sequential detection prefetching to read ahead prefetching whenever low density indexes exist.

  If table data was initially clustered on a certain index, and the clustering statistics indicate that the data is now poorly clustered on that same index, you might want to reorganize the table to re-cluster the data. Also, if smart data prefetching is enabled, it can improve the performance of poorly clustered data which reduces the need to perform the **REORG** command on tables. Smart data prefetching achieves this by switching from sequential detection prefetching to read ahead prefetching whenever badly clustered data pages exist.

- Number of leaf pages

  Query the NLEAF column in the SYSCAT.INDEXES view to determine the number of leaf pages that are occupied by an index. This number tells you how many index page I/Os are needed for a complete scan of the index.

  Ideally, an index should occupy as little space as possible to reduce the number of I/Os that are required for an index scan. Random update activity can cause page splits that increase the size of an index. During a table **REORG** operation, each index can be rebuilt with the least amount of space.

  By default, ten percent of free space is left on each index page when an index is built. To increase the free space amount, specify the PCTFREE option when you create the index. The specified PCTFREE value is used whenever you reorganize the index. A free space value that is greater than ten percent might reduce the frequency of index reorganization, because the extra space can accommodate additional index insertions.

- Number of empty data pages

  To calculate the number of empty pages in a table, query the FPAGES and NPAGES columns in the SYSCAT.TABLES view and then subtract the NPAGES value (the number of pages that contain rows) from the FPAGES value (the total number of pages in use). Empty pages can occur when entire ranges of rows are deleted.

  As the number of empty pages increases, so does the need for table reorganization. Reorganizing a table reclaims empty pages and reduces the amount of space that a table uses. In addition, because empty pages are read into the buffer pool during a table scan, reclaiming unused pages can improve scan performance.

  If the total number of in-use pages (FPAGES) in a table is less than or equal to (NPARTITIONS * 1 extent size), table reorganization is not recommended. NPARTITIONS represents the number of data partitions if the table is a partitioned table; otherwise, its value is 1. In a partitioned database environment, table reorganization is not recommended if FPAGES <= (the number of database partitions in a database partition group of the table) * (NPARTITIONS * 1 extent size).

Before reorganizing tables or indexes, consider the trade-off between the cost of increasingly degraded query performance and the cost of table or index reorganization, which includes processing time, elapsed time, and reduced concurrency.

Costs of table and index reorganization

Performing a table reorganization or an index reorganization with the **REBUILD** option incurs a certain amount of overhead that must be considered when deciding whether to reorganize an object.

The costs of reorganizing tables and reorganizing indexes with the **REBUILD** option include:

- Processing time of the executing utility
- Reduced concurrency (because of locking) while running the reorg utility.
- Extra storage requirements.
  - Offline table reorganization requires more storage space to hold a shadow copy of the table.
  - Online or inplace table reorganization requires more log space.
  - Offline index reorganization requires less log space and does not involve a shadow copy.
  - Online index reorganization requires more log space and more storage space to hold a shadow copy of the index.

In some cases, a reorganized table might be larger than the original table. A table might grow after reorganization in the following situations:

- In a clustering reorg table operation in which an index is used to determine the order of the rows, more space might be required if the table records are of a variable length, because some pages in the reorganized table might contain fewer rows than in the original table.

- The amount of free space left on each page (represented by the PCTFREE value) might have increased since the last reorganization.

Space requirements for an offline table reorganization

Because offline reorganization uses a shadow copy approach, you need enough additional storage to accommodate another copy of the table. The shadow copy is built either in the table space in which the original table resides or in a user-specified temporary table space.

Additional temporary table space storage might be required for sort processing if a table scan sort is used. The additional space required might be as large as the size of the table being reorganized. If the clustering index is of system managed space (SMS) type or unique database managed space (DMS) type, the recreation of this index does not require a sort. Instead, this index is rebuilt by scanning the newly reorganized data. Any other indexes that are recreated will require a sort, potentially involving temporary space up to the size of the table being reorganized.

Offline table reorg operations generate few control log records, and therefore consume a relatively small amount of log space. If the reorg utility does not use an index, only table data log records are created. If an index is specified, or if there is a clustering index on the table, record IDs (RIDs) are logged in the order in which they are placed into the new version of the table. Each RID log record holds a maximum of 8000 RIDs, with each RID consuming 4 bytes. This can contribute to log space problems during an offline table reorg operation. Note that RIDs are only logged if the database is recoverable.

Log space requirements for an online table reorganization

The log space that is required for an online table reorg operation is typically larger than what is required for an offline table reorg. The amount of space that is required is determined by the number of rows being reorganized, the number of indexes, the size of the index keys, and how poorly organized the table is at the outset. It is a good idea to establish a typical benchmark for log space consumption associated with your tables.

Every row in a table is likely moved twice during an online table reorg operation. For each index, each table row must update the index key to reflect the new location, and after all accesses to the old location have completed, the index key is updated again to remove references to the old RID. When the row is moved back, updates to the index key are performed again. All of this activity is logged to make online table reorganization fully recoverable. There is a minimum of two data log records (each including the row data) and four index log records (each including the key data) for each row (assuming one index). Clustering indexes, in particular, are prone to filling up the index pages, causing index splits and merges which must also be logged.

Because the online table reorg utility issues frequent internal COMMIT statements, it usually does not hold a large number of active logs. An exception can occur during the truncation phase, when the utility requests an S table lock. If the utility cannot acquire the lock, it waits, and other transactions might quickly fill up the logs in the meantime.

Reducing the need to reorganize tables and indexes

You can use different strategies to reduce the need for (and the costs associated with) table and index reorganization.

Reducing the need to reorganize tables

To reduce the need for table reorganization:

- Use multi-partition tables.
- Create multidimensional clustering (MDC) tables. For MDC tables, clustering is maintained on the columns that you specify with the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. However, the reorgchk utility might still recommend reorganization of an MDC table if it determines that there are too many unused blocks or that blocks should be compacted.
- Create insert time clustering (ITC) tables. For ITC tables, if you have a cyclical access pattern, for example you delete all data that was inserted at similar times, you can release that space back to the system. In such cases, you can reduce the need for a table reorganization with the REORG RECLAIM EXTENTS command that frees space.
- Enable the APPEND mode on your tables. If the index key values for new rows are always new high key values, for example, the clustering attribute of the table will attempt to place them at the end of the table. In this case, enabling the APPEND mode might be a better choice than using a clustering index.

To further reduce the need for table reorganization, perform these tasks after you create a table:

- Alter the table to specify the percentage of each page that is to be left as free space during a load or a table reorganization operation (PCTFREE)
- Create a clustering index, specifying the PCTFREE option
- Sort the data before loading it into the table

After you have performed these tasks, the clustering index and the PCTFREE setting on the table help to preserve the original sorted order. If there is enough space on the table pages, new data can be inserted on the correct pages to maintain the clustering characteristics of the index. However, as more data is inserted and the table pages become full, records are appended to the end of the table, which gradually becomes unclustered.

If you perform a table reorg operation or a sort and load operation after you create a clustering index, the index attempts to maintain the order of the data, which improves the CLUSTERRATIO or CLUSTERFACTOR statistics that are collected by the runstats utility.

Note: If readahead prefetching is enabled it helps reduce the need to reorganize tables even if formula F4 of the **REORGCHK** command states otherwise.

Reducing the need to rebuild indexes

To reduce the need to rebuild indexes with index reorganization:

- Create indexes specifying the PCTFREE or the LEVEL2 PCTFREE option.
- Create indexes with the MINPCTUSED option. Alternatively, consider using the CLEANUP ALL option of the REORG INDEXES command to merge leaf pages.

- Use the RECLAIM EXTENTS option of the REORG INDEXES command to release space back to the table space in an online fashion. This operation provides space reclaim without the need for a full rebuild of the indexes.

Note: If readahead prefetching is enabled it helps reduce the need to rebuild indexes with index reorganization, even if formula F4 of the **REORGCHK** command states otherwise.

Enabling automatic table and index reorganization

Use automatic table and index reorganization to eliminate the worry of when and how to reorganize your data.

About this task

Having well-organized table and index data is critical to efficient data access and optimal workload performance. After many database operations, such as insert, update, and delete, logically sequential table data might be found on nonsequential data pages. When logically sequential table data is found on nonsequential data pages, additional read operations are required by the database manager to access data. Additional read operations are also required when accessing data in a table from which a significant number of rows are deleted. You can enable the database manager to reorganize system both catalog tables and user tables.

Procedure

To enable your database for automatic reorganization:

1. Set the auto_maint, auto_tbl_maint, and auto_reorg database configuration parameters to ON. You can set the parameters to ON with these commands:
   - `db2 update db cfg for <db_name> using auto_maint on`
   - `db2 update db cfg for <db_name> using auto_tbl_maint on`
   - `db2 update db cfg for <db_name> using auto_reorg on`

   Replace <db_name> with the name of the database on which you want to enable automatic maintenance and reorganization.
2. Connect to the database, <db_name>.
3. Specify a reorganization policy. A reorganization policy is a defined set of rules or guidelines that dictate when automated table and index maintenance takes place. You can set this policy in one of two ways:
   a. Call the AUTOMAINT_SET_POLICY procedure.
   b. Call the AUTOMAINT_SET_POLICYFILE procedure.

   The reorganization policy is either an input argument or file both of which are in an XML format. For more information about both of these procedures, see "Automatic table and index maintenance" on page 51.

Enabling automatic index reorganization in volatile tables

You can enable automatic reorganization to perform index reorganization in volatile tables.

About this task

If you enable automatic index reorganization in volatile tables, automatic reorg checks at every refresh interval whether the indexes on volatile tables require reorganization and schedules the necessary operation using the **REORG** command.

Procedure

To enable automatic index reorganization in volatile tables, perform the following steps:

1. Set the **DB2_WORKLOAD** registry variable to SAP. The following example shows how to set this variable using the **db2set** command:

   ```
   db2set DB2_WORKLOAD=SAP
   ```

   Restart the database so that this setting takes effect.
2. Set the **auto_reorg** database configuration parameter to ON. The following example shows how to set this database configuration parameter using the DB2 CLP command line interface:

   ```
   UPDATE DB CFG FOR SAMPLE USING auto_reorg ON
   ```

   Ensure that the **auto_maint** and **auto_tbl_maint** database configuration parameters are also set to ON. By the default, **auto_maint** and **auto_tbl_maint** are set to ON.
3. Set the numInxPseudoEmptyPagesForVolatileTables attribute in the AUTO_REORG policy by calling the AUTOMAINT_SET_POLICY or AUTOMAINT_SET_POLICYFILE procedure. This attribute indicates the minimum number of empty index pages with pseudo deleted keys required to perform the index reorganization. The following example shows how to set this attribute:

   ```
   CALL SYSPROC.AUTOMAINT_SET_POLICY
     ('AUTO_REORG',
     BLOB(' <?xml version="1.0" encoding="UTF-8"?>
   <DB2AutoReorgPolicy
   xmlns="http://www.ibm.com/xmlns/prod/db2/autonomic/config" >

    <ReorgOptions  dictionaryOption="Keep" indexReorgMode="Online"
       useSystemTempTableSpace="false"
    numInxPseudoEmptyPagesForVolatileTables="20" />

    <ReorgTableScope  maxOfflineReorgTableSize="0">
     <FilterClause>TABSCHEMA NOT LIKE 'SYS%'</FilterClause>
    </ReorgTableScope>
   </DB2AutoReorgPolicy>')
     )
   ```

   You can monitor the values for the PSEUDO_EMPTY_PAGES, EMPTY_PAGES_DELETED, and EMPTY_PAGES_REUSED column by querying the MON_GET_INDEX table function to help you determine an appropriate value for the numInxPseudoEmptyPagesForVolatileTables attribute.

Chapter 40. Catalog statistics

When the query compiler optimizes query plans, its decisions are heavily influenced by statistical information about the size of the database tables, indexes, and statistical views. This information is stored in system catalog tables.

The optimizer also uses information about the distribution of data in specific columns of tables, indexes, and statistical views if these columns are used to select rows or to join tables. The optimizer uses this information to estimate the costs of alternative access plans for each query.

Statistical information about the cluster ratio of indexes, the number of leaf pages in indexes, the number of table rows that overflow their original pages, and the number of filled and empty pages in a table can also be collected. You can use this information to decide when to reorganize tables or indexes.

Table statistics in a partitioned database environment are collected only for that portion of the table that resides on the database partition on which the utility is running, or for the first database partition in the database partition group that contains the table. Information about statistical views is collected for all database partitions.

Statistics that are updated by the runstats utility

Catalog statistics are collected by the runstats utility, which can be started by issuing the **RUNSTATS** command, calling the ADMIN_CMD procedure, or calling the db2Runstats API. Updates can be initiated either manually or automatically.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for collecting catalog statistics. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Statistics about declared temporary tables are not stored in the system catalog, but are stored in memory structures that represent the catalog information for declared temporary tables. It is possible (and in some cases, it might be useful) to perform runstats on a declared temporary table.

The runstats utility collects the following information about tables and indexes:

- The number of pages that contain rows
- The number of pages that are in use
- The number of rows in the table (the *cardinality*)
- The number of rows that overflow
- For multidimensional clustering (MDC) and insert time clustering (ITC) tables, the number of blocks that contain data
- For partitioned tables, the degree of data clustering within a single data partition
- Data distribution statistics, which are used by the optimizer to estimate efficient access plans for tables and statistical views whose data is not evenly distributed and whose columns have a significant number of duplicate values

- Detailed index statistics, which are used by the optimizer to determine how efficient it is to access table data through an index
- Subelement statistics for LIKE predicates, especially those that search for patterns within strings (for example, `LIKE %disk%`), are also used by the optimizer

The runstats utility collects the following statistics for each data partition in a table. These statistics are only used for determining whether a partition needs to be reorganized:
- The number of pages that contain rows
- The number of pages that are in use
- The number of rows in the table (the cardinality)
- The number of rows that overflow
- For MDC and ITC tables, the number of blocks that contain data

Distribution statistics are not collected:
- When the **num_freqvalues** and **num_quantiles** database configuration parameters are set to 0
- When the distribution of data is known, such as when each data value is unique
- When the column contains a LONG, LOB, or structured data type
- For row types in sub-tables (the table-level statistics NPAGES, FPAGES, and OVERFLOW are not collected)
- If quantile distributions are requested, but there is only one non-null value in the column
- For extended indexes or declared temporary tables

The runstats utility collects the following information about each column in a table or statistical view, and the first column in an index key:
- The cardinality of the column
- The average length of the column (the average space, in bytes, that is required when the column is stored in database memory or in a temporary table)
- The second highest value in the column
- The second lowest value in the column
- The number of null values in the column

For columns that contain large object (LOB) or LONG data types, the runstats utility collects only the average length of the column and the number of null values in the column. The average length of the column represents the length of the data descriptor, except when LOB data is located inline on the data page. The average amount of space that is required to store the column on disk might be different than the value of this statistic.

The runstats utility collects the following information about each XML column:
- The number of NULL XML documents
- The number of non-NULL XML documents
- The number of distinct paths
- The sum of the node count for each distinct path
- The sum of the document count for each distinct path
- The *k* pairs of (path, node count) with the largest node count
- The *k* pairs of (path, document count) with the largest document count

- The *k* triples of (path, value, node count) with the largest node count
- The *k* triples of (path, value, document count) with the largest document count
- For each distinct path that leads to a text or attribute value:
  - The number of distinct values that this path can take
  - The highest value
  - The lowest value
  - The number of text or attribute nodes
  - The number of documents that contain the text or attribute nodes

Each row in an XML column stores an XML document. The node count for a path or path-value pair refers to the number of nodes that are reachable by that path or path-value pair. The document count for a path or path-value pair refers to the number of documents that contain that path or path-value pair.

For DB2 V9.7 Fix Pack 1 and later releases, the following apply to the collection of distribution statistics on an XML column:

- Distribution statistics are collected for each index over XML data specified on an XML column.
- The runstats utility must collect both distribution statistics and table statistics to collect distribution statistics for an index over XML data. Table statistics must be gathered in order for distribution statistics to be collected since XML distribution statistics are stored with table statistics.

  Collecting only index statistics, or collecting index statistics during index creation, will not collect distribution statistics for an index over XML data.

  As the default, the runstats utility collects a maximum of 250 quantiles for distribution statistics for each index over XML data. The maximum number of quantiles for a column can be specified when executing the runstats utility.
- Distribution statistics are collected for indexes over XML data of type VARCHAR, DOUBLE, TIMESTAMP, and DATE. XML distribution statistics are not collected for indexes over XML data of type VARCHAR HASHED.
- XML distribution statistics are collected when automatic table runstats operations are performed.
- XML distribution statistics are not created when loading data with the STATISTICS option.
- XML distribution statistics are not collected for partitioned indexes over XML data defined on a partitioned table.

The runstats utility collects the following information about column groups:

- A timestamp-based name for the column group
- The cardinality of the column group

The runstats utility collects the following information about indexes:

- The number of index entries (the *index cardinality*)
- The number of leaf pages
- The number of index levels
- The degree of clustering of the table data to the index
- The degree of clustering of the index keys with regard to data partitions
- The ratio of leaf pages located on disk in index key order to the number of pages in the range of pages occupied by the index
- The number of distinct values in the first column of the index

- The number of distinct values in the first two, three, and four columns of the index
- The number of distinct values in all columns of the index
- The number of leaf pages located on disk in index key order, with few or no large gaps between them
- The average leaf key size, without include columns
- The average leaf key size, with include columns
- The number of pages on which all record identifiers (RIDs) are marked deleted
- The number of RIDs that are marked deleted on pages where not all RIDs are marked deleted

If you request detailed index statistics, additional information about the degree of clustering of the table data to the index, and the page fetch estimates for different buffer sizes, is collected.

For a partitioned index, these statistics are representative of a single index partition, with the exception of the distinct values in the first column of the index; the first two, three, and four columns of the index; and in all columns of the index. Per-index partition statistics are also collected for the purpose of determining whether an index partition needs to be reorganized.

Statistics collection invalidates cached dynamic statements that reference tables for which statistics have been collected. This is done so that cached dynamic statements can be re-optimized with the latest statistics.

Catalog statistics tables

Statistical information about the size of database tables, indexes, and statistical views is stored in system catalog tables.

Catalog statistics views

The database manager creates and maintains catalog views that are defined on top of the base system catalog tables. A set of updatable catalog views are created under the SYSSTAT schema. These updatable views contain statistical information that is used by the optimizer. The values in some columns in these views can be changed to test performance. Before any statistics are changed, it is recommended that you issue the RUNSTATS command so that all the statistics reflect the current state. Applications should use the SYSSTAT views rather than the base catalog tables.

Guidelines for collecting and updating statistics

The **RUNSTATS** utility collects statistics on tables, indexes, and statistical views to provide the optimizer with accurate information for access plan selection.

Use the **RUNSTATS** utility to collect statistics in the following situations:

- After data is loaded into a table and appropriate indexes are created
- After creating an index on a table
- After a table is reorganized with the **REORG** utility
- After a table and its indexes are significantly modified through update, insert, or delete operations
- Before binding application programs whose performance is critical

- When you want to compare current and previous statistics
- When the prefetch value has changed
- After executing the **REDISTRIBUTE DATABASE PARTITION GROUP** command
- When you have XML columns. When **RUNSTATS** is used to collect statistics for XML columns only, existing statistics for non-XML columns that were collected during a load operation or a previous **RUNSTATS** operation are retained. If statistics on some XML columns were collected previously, those statistics are either replaced or dropped if the current **RUNSTATS** operation does not include those columns.

To improve **RUNSTATS** performance and save disk space used to store statistics, consider specifying only those columns for which data distribution statistics should be collected.

You should rebind application programs after executing **RUNSTATS**. The query optimizer might choose different access plans if new statistics are available.

If a full set of statistics cannot be collected at one time, use the **RUNSTATS** utility on subsets of the objects. If inconsistencies occur as a result of ongoing activity against those objects, a warning message (SQL0437W, reason code 6) is returned during query optimization. If this occurs, use **RUNSTATS** again to update the distribution statistics.

To ensure that index statistics are synchronized with the corresponding table, collect both table and index statistics at the same time. If a table was modified extensively since the last time that statistics were gathered, updating only the index statistics for that table leaves the two sets of statistics out of synchronization with each other.

Using the **RUNSTATS** utility on a production system might negatively affect workload performance. The utility now supports a throttling option that can be used to limit the performance impact of **RUNSTATS** execution during high levels of database activity.

When you collect statistics for a table in a partitioned database environment, **RUNSTATS** operates only on the database partition from which the utility is executed. The results from this database partition are extrapolated to the other database partitions. If this database partition does not contain a required portion of the table, the request is sent to the first database partition in the database partition group that contains the required data.

Statistics for a statistical view are collected on all database partitions containing base tables that are referenced by the view.

Consider the following tips to improve the efficiency of **RUNSTATS** and the usefulness of the statistics:

- Collect statistics only for columns that are used to join tables or for columns that are referenced in the WHERE, GROUP BY, or similar clauses of queries. If the columns are indexed, you can specify these columns with the **ONLY ON KEY COLUMNS** clause on the **RUNSTATS** command.
- Customize the values of the **num_freqvalues** and **num_quantiles** database configuration parameters for specific tables and columns.
- When you create an index for a populated table, use the COLLECT STATISTICS clause to create statistics as the index is created.

- When significant numbers of table rows are added or removed, or if data in columns for which you collect statistics is updated, use **RUNSTATS** again to update the statistics.
- Because **RUNSTATS** collects statistics on only a single database partition, the statistics are less accurate if the data is not distributed consistently across all database partitions. If you suspect that there is skewed data distribution, consider redistributing the data across database partitions by using the **REDISTRIBUTE DATABASE PARTITION GROUP** command before using the **RUNSTATS** utility.
- For DB2 V9.7 Fix Pack 1 and later releases, distribution statistics can be collected on an XML column. Distribution statistics are collected for each index over XML data specified on the XML column. By default, a maximum of 250 quantiles are used for distribution statistics for each index over XML data.

  When collecting distribution statistics on an XML column, you can change maximum number of quantiles. You can lower the maximum number of quantiles to reduce the space requirements for XML distribution statistics based on your particular data size, or you can increase the maximum number of quantiles if 250 quantiles are not sufficient to capture the distribution statistics of the data set for an index over XML data.

Detailed index statistics

A **RUNSTATS** operation for indexes with the **DETAILED** parameter collects statistical information that allows the optimizer to estimate how many data page fetches are required, depending on the buffer pool size. This additional information helps the optimizer to better estimate the cost of accessing a table through an index.

Detailed statistics provide concise information about the number of physical I/Os that are required to access the data pages of a table if a complete index scan is performed under different buffer pool sizes. As the **RUNSTATS** utility scans the pages of an index, it models the different buffer sizes, and estimates how often a page fault occurs. For example, if only one buffer page is available, each new page that is referenced by the index results in a page fault. In the worst case, each row might reference a different page, resulting in at most the same number of I/Os as the number of rows in the indexed table. At the other extreme, when the buffer is big enough to hold the entire table (subject to the maximum buffer size), all table pages are read at once. As a result, the number of physical I/Os is a monotonic, nonincreasing function of the buffer size.

The statistical information also provides finer estimates of the degree of clustering of the table rows to the index order. The less clustering, the more I/Os are required to access table rows through the index. The optimizer considers both the buffer size and the degree of clustering when it estimates the cost of accessing a table through an index.

Collect detailed index statistics when:

- Queries reference columns that are not included in the index
- The table has multiple non-clustered indexes with varying degrees of clustering
- The degree of clustering among the key values is nonuniform
- Index values are updated in a nonuniform manner

It is difficult to identify these conditions without previous knowledge or without forcing an index scan under varying buffer sizes and then monitoring the resulting physical I/Os. Perhaps the least expensive way to determine whether any of these conditions exist is to collect and examine the detailed statistics for an index, and to retain them if the resulting PAGE_FETCH_PAIRS are nonlinear.

When you collect detailed index statistics, the **RUNSTATS** operation takes longer to complete and requires more memory and processing time. The **DETAILED** option (equivalent to the **SAMPLED DETAILED** parameter), for example, requires 2 MB of the statistics heap. Allocate an additional 488 4-KB pages to the **stat_heap_sz** database configuration parameter setting for this memory requirement. If the heap is too small, the **RUNSTATS** utility returns an error before it attempts to collect statistics.

CLUSTERFACTOR and PAGE_FETCH_PAIRS are not collected unless the table is of sufficient size (greater than about 25 pages). In this case, CLUSTERFACTOR will be a value between 0 and 1, and CLUSTERRATIO is -1 (not collected). If the table is relatively small, only CLUSTERRATIO, with a value between 0 and 100, is collected by the **RUNSTATS** utility; CLUSTERFACTOR and PAGE_FETCH_PAIRS are not collected. If the **DETAILED** clause is not specified, only CLUSTERRATIO is collected.

Distribution statistics

You can collect two kinds of data distribution statistics: frequent-value statistics and quantile statistics.

- *Frequent-value statistics* provide information about a column and the data value with the highest number of duplicates, the value with the second highest number of duplicates, and so on, to the level that is specified by the value of the **num_freqvalues** database configuration parameter. To disable the collection of frequent-value statistics, set **num_freqvalues** to 0. You can also use the **NUM_FREQVALUES** clause on the **RUNSTATS** command for a specific table, statistical view, or column.
- *Quantile statistics* provide information about how data values are distributed in relation to other values. Called *K*-quantiles, these statistics represent the value *V* at or below which at least *K* values lie. You can compute a *K*-quantile by sorting the values in ascending order. The *K*-quantile value is the value in the *K*th position from the low end of the range.

  To specify the number of "sections" (quantiles) into which the column data values should be grouped, set the **num_quantiles** database configuration parameter to a value between **2** and **32 767**. The default value is **20**, which ensures a maximum optimizer estimation error of plus or minus 2.5% for any equality, less-than, or greater-than predicate, and a maximum error of plus or minus 5% for any BETWEEN predicate. To disable the collection of quantile statistics, set **num_quantiles** to **0** or **1**.

  You can set **num_quantiles** for a specific table, statistical view, or column.

Note: The **RUNSTATS** utility consumes more processing resources and memory (specified by the **stat_heap_sz** database configuration parameter) if larger **num_freqvalues** and **num_quantiles** values are used.

When to collect distribution statistics

To decide whether distribution statistics for a table or statistical view would be helpful, first determine:

- Whether the queries in an application use host variables.

Distribution statistics are most useful for dynamic and static queries that do not use host variables. The optimizer makes limited use of distribution statistics when assessing queries that contain host variables.

- Whether the data in columns is uniformly distributed.

  Create distribution statistics if at least one column in the table has a highly "nonuniform" distribution of data, and the column appears frequently in equality or range predicates; that is, in clauses such as the following:

  ```
  where c1 = key;
  where c1 in (key1, key2, key3);
  where (c1 = key1) or (c1 = key2) or (c1 = key3);
  where c1 <= key;
  where c1 between key1 and key2;
  ```

Two types of nonuniform data distribution can occur, and possibly together.

- Data might be highly clustered instead of being evenly spread out between the highest and lowest data value. Consider the following column, in which the data is clustered in the range (5,10):

  ```
    0.0
    5.1
    6.3
    7.1
    8.2
    8.4
    8.5
    9.1
   93.6
  100.0
  ```

  Quantile statistics help the optimizer to deal with this kind of data distribution.

  Queries can help you to determine whether column data is not uniformly distributed. For example:

  ```
  select c1, count(*) as occurrences
    from t1
  group by c1
  order by occurrences desc
  ```

- Duplicate data values might often occur. Consider a column in which the data is distributed with the following frequencies:

Table 119. Frequency of data values in a column

Data Value	Frequency
20	5
30	10
40	10
50	25
60	25
70	20
80	5

  Both frequent-value and quantile statistics help the optimizer to deal with numerous duplicate values.

When to collect index statistics only

You might consider collecting statistics that are based only on index data in the following situations:

- A new index was created since the **RUNSTATS** utility was run, and you do not want to collect statistics again on the table data.
- There were many changes to the data that affect the first column of an index.

What level of statistical precision to specify

Use the **num_quantiles** and **num_freqvalues** database configuration parameters to specify the precision with which distribution statistics are stored. You can also specify the precision with corresponding **RUNSTATS** command options when you collect statistics for a table or for columns. The higher you set these values, the greater the precision that the **RUNSTATS** utility uses when it creates and updates distribution statistics. However, greater precision requires more resources, both during the **RUNSTATS** operation itself, and for storing more data in the catalog tables.

For most databases, specify between 10 and 100 as the value of the **num_freqvalues** database configuration parameter. Ideally, frequent-value statistics should be created in such a way that the frequencies of the remaining values are either approximately equal to one another or negligible when compared to the frequencies of the most frequent values. The database manager might collect fewer than this number, because these statistics are collected only for data values that occur more than once. If you need to collect only quantile statistics, set the value of **num_freqvalues** to zero.

To specify the number of quantiles, set the **num_quantiles** database configuration parameter to a value between 20 and 50.
- First determine the maximum acceptable error when estimating the number of rows for any range query, as a percentage *P*.
- The number of quantiles should be approximately 100/*P* for BETWEEN predicates, and 50/*P* for any other type of range predicate (<, <=, >, or >=).

For example, 25 quantiles should result in a maximum estimate error of 4% for BETWEEN predicates and 2% for ">" predicates. In general, specify at least 10 quantiles. More than 50 quantiles should be necessary only for extremely nonuniform data. If you need only frequent-value statistics, set **num_quantiles** to 0. If you set this parameter to 1, because the entire range of values fits within one quantile, no quantile statistics are collected.

Optimizer use of distribution statistics

The optimizer uses distribution statistics for better estimates of the cost of different query access plans.

Unless it has additional information about the distribution of values between the low and high values, the optimizer assumes that data values are evenly distributed. If data values differ widely from each other, are clustered in some parts of the range, or contain many duplicate values, the optimizer will choose a less than optimal access plan.

Consider the following example: To select the least expensive access plan, the optimizer needs to estimate the number of rows with a column value that satisfies an equality or range predicate. The more accurate the estimate, the greater the likelihood that the optimizer will choose the optimal access plan. For the following query:

```
select c1, c2
  from table1
  where c1 = 'NEW YORK'
  and c2 <= 10
```

Assume that there is an index on both columns C1 and C2. One possible access plan is to use the index on C1 to retrieve all rows with `C1 = 'NEW YORK'`, and then to check whether `C2 <= 10` for each retrieved row. An alternate plan is to use the index on C2 to retrieve all rows with `C2 <= 10`, and then to check whether `C1 = 'NEW YORK'` for each retrieved row. Because the primary cost of executing a query is usually the cost of retrieving the rows, the best plan is the one that requires the fewest retrievals. Choosing this plan means estimating the number of rows that satisfy each predicate.

When distribution statistics are not available, but the runstats utility has been used on a table or a statistical view, the only information that is available to the optimizer is the second-highest data value (HIGH2KEY), the second-lowest data value (LOW2KEY), the number of distinct values (COLCARD), and the number of rows (CARD) in a column. The number of rows that satisfy an equality or range predicate is estimated under the assumption that the data values in the column have equal frequencies and that the data values are evenly distributed between LOW2KEY and HIGH2KEY. Specifically, the number of rows that satisfy an equality predicate (`C1 = KEY`) is estimated as CARD/COLCARD, and the number of rows that satisfy a range predicate (`C1 BETWEEN KEY1 AND KEY2`) can be estimated with the following formula:

```
      KEY2 - KEY1
------------------  x CARD
HIGH2KEY - LOW2KEY
```

These estimates are accurate only when the true distribution of data values within a column is reasonably uniform. When distribution statistics are unavailable, and either the frequency of data values varies widely, or the data values are very unevenly distributed, the estimates can be off by orders of magnitude, and the optimizer might choose a suboptimal access plan.

When distribution statistics are available, the probability of such errors can be greatly reduced by using frequent-value statistics to estimate the number of rows that satisfy an equality predicate, and by using both frequent-value statistics and quantile statistics to estimate the number of rows that satisfy a range predicate.

Enabling automatic statistics collection

Having accurate and complete database statistics is critical to efficient data access and optimal workload performance. Use the automatic statistics collection feature of the automated table maintenance functionality to update and maintain relevant database statistics.

About this task

You can enhance this functionality in environments where a single database partition operates on a single processor by collecting query data and generating statistics profiles that help the DB2 server to automatically collect the exact set of statistics that is required by your workload. This option is not available in partitioned database environments, certain federated database environments, or environments in which intrapartition parallelism is enabled.

To enable automatic statistics collection, you must first configure your database by setting the **auto_maint** and the **auto_tbl_maint** database configuration parameters to ON.

Procedure

After setting the **auto_maint** and the **auto_tbl_maint** database configuration parameters to ON, you have the following options:

- To enable background statistics collection, set the **auto_runstats** database configuration parameter to ON.
- To enable background statistics collection for statistical views, set both the **auto_stats_views** and **auto_runstats** database configuration parameters to ON.
- To enable background statistics collection to use sampling automatically for large tables and statistical views, also set the **auto_sampling** database configuration parameter to ON. Use this setting in addition to **auto_runstats** (tables only) or to **auto_runstats** and **auto_stats_views** (tables and statistical views).
- To enable real-time statistics collection, set both **auto_stmt_stats** and **auto_runstats** database configuration parameters to ON.

Collecting statistics using a statistics profile

The **RUNSTATS** utility provides the option to register and use a statistics profile, which specifies the type of statistics that are to be collected for a particular table; for example, table statistics, index statistics, or distribution statistics. This feature simplifies statistics collection by enabling you to store **RUNSTATS** options for convenient future use.

To register a profile and collect statistics at the same time, issue the **RUNSTATS** command with the **SET PROFILE** parameter. To register a profile only, issue the **RUNSTATS** command with the **SET PROFILE ONLY** parameter. To collect statistics using a profile that was already registered, issue the **RUNSTATS** command with the **USE PROFILE** parameter.

To see what options are currently specified in the statistics profile for a particular table, query the SYSCAT.TABLES catalog view. For example:

```
SELECT STATISTICS_PROFILE FROM SYSCAT.TABLES WHERE TABNAME = 'EMPLOYEE'
```

Automatic statistics profiling

Statistics profiles can also be generated automatically with the DB2 automatic statistics profiling feature. When this feature is enabled, information about database activity is collected and stored in the query feedback warehouse. A statistics profile is then generated based on this data. Enabling this feature can alleviate the uncertainty about which statistics are relevant to a particular workload.

Important: Automatic statistics profiling is deprecated in Version 10.1 and might be removed in a future release. For more information, see “Automatic statistics profiling is deprecated” in *What's New for DB2 Version 10.1*.

Automatic statistics profiling can be used with automatic statistics collection, which schedules statistics maintenance operations based on information contained in the automatically generated statistics profile.

To enable automatic statistics profiling, ensure that automatic table maintenance was already enabled by setting the appropriate database configuration parameters. For more information, see “auto_maint - Automatic maintenance configuration parameter”. The **auto_stats_prof** configuration parameter activates the collection of query feedback data, and the **auto_prof_upd** configuration parameter activates the generation of a statistics profile for use by automatic statistics collection.

Automatic statistics profile generation is not supported in partitioned database environments, in certain federated database environments, in DB2 pureScale environments, or when intra-partition parallelism is enabled. Automatic statistics profile generation cannot be enabled if the **section_actuals** database configuration parameter is enabled (SQLCODE -5153).

Automatic statistics profiling is best suited to systems running large complex queries that have many predicates, use large joins, or specify extensive grouping. It is less suited to systems with primarily transactional workloads.

In a development environment, where the performance overhead of runtime monitoring can easily be tolerated, set the **auto_stats_prof** and **auto_prof_upd**configuration parameters to ON. When a test system uses realistic data and queries, appropriate statistics profiles can be transferred to the production system, where queries can benefit without incurring additional monitoring overhead.

In a production environment, if performance problems with a particular set of queries (problems that can be attributed to faulty statistics) are detected, you can set the**auto_stats_prof** configuration parameter to ON and execute the target workload for a period of time. Automatic statistics profiling analyzes the query feedback and create recommendations in the SYSTOOLS.OPT_FEEDBACK_RANKING tables. You can inspect these recommendations and refine the statistics profiles manually, as appropriate. To have the DB2 server automatically update the statistics profiles based on these recommendations, enable **auto_prof_upd** when you enable **auto_stats_prof**.

Creating the query feedback warehouse

The query feedback warehouse, which is required for automatic statistics profiling, consists of five tables in the SYSTOOLS schema. These tables store information about the predicates that are encountered during query execution, as well as recommendations for statistics collection. The five tables are:

- OPT_FEEDBACK_PREDICATE
- OPT_FEEDBACK_PREDICATE_COLUMN
- OPT_FEEDBACK_QUERY
- OPT_FEEDBACK_RANKING
- OPT_FEEDBACK_RANKING_COLUMN

Use the SYSINSTALLOBJECTS procedure to create the query feedback warehouse. For more information about this procedure, which is used to create or drop objects in the SYSTOOLS schema, see “SYSINSTALLOBJECTS”.

Storage used by automatic statistics collection and profiling

The automatic statistics collection and reorganization features store working data in tables that are part of your database. These tables are created in the SYSTOOLSPACE table space.

SYSTOOLSPACE is created automatically with default options when the database is activated. Storage requirements for these tables are proportional to the number of tables in the database and can be estimated at approximately 1 KB per table. If this is a significant size for your database, you might want to drop and then recreate the table space yourself, allocating storage appropriately. Although the automatic maintenance and health monitoring tables in the table space are automatically recreated, any history that was captured in those tables is lost when you drop the table space.

Automatic statistics collection activity logging

The statistics log is a record of all of the statistics collection activities (both manual and automatic) that occurred against a specific database. The default name of the statistics log is db2optstats.number.log. It is stored in the $diagpath/events directory. The statistics log is a rotating log. Log behavior is controlled by the DB2_OPTSTATS_LOG registry variable. The statistics log can be viewed directly or it can be queried by using the SYSPROC.PD_GET_DIAG_HIST table function.

Improving query performance for large statistics logs

If the statistics log files are large, you can improve query performance by copying the log records into a table, creating indexes, and then gathering statistics.

Procedure

1. Create a table with appropriate columns for the log records.

```
create table db2user.stats_log (
  pid         bigint,
  tid         bigint,
  timestamp   timestamp,
  dbname      varchar(128),
  retcode     integer,
  eventtype   varchar(24),
  objtype     varchar(30),
  objschema   varchar(20),
  objname     varchar(30),
  event1_type varchar(20),
  event1      timestamp,
  event2_type varchar(20),
  event2      varchar(40),
  event3_type varchar(20),
  event3      varchar(40),
  eventstate  varchar(20))
```

2. Declare a cursor for a query against SYSPROC.PD_GET_DIAG_HIST.

```
declare c1 cursor for
  select pid, tid, timestamp, dbname, retcode, eventtype,
      substr(objtype, 1, 30) as objtype,
      substr(objname_qualifier, 1, 20) as objschema,
      substr(objname, 1, 30) as objname,
      substr(first_eventqualifiertype, 1, 20),
      substr(first_eventqualifier, 1, 26),
      substr(second_eventqualifiertype, 1, 20),
      substr(second_eventqualifier, 1, 40),
      substr(third_eventqualifiertype, 1, 20),
      substr(third_eventqualifier, 1, 40),
      substr(eventstate, 1, 20)
    from table (sysproc.pd_get_diag_hist
      ('optstats', 'EX', 'NONE',
        current_timestamp - 1 year, cast(null as timestamp ))) as sl
```

3. Load the statistics log records into the table.

```
load from c1 of cursor replace into db2user.stats_log
```

4. Create indexes and then gather statistics on the table.

   ```
   create index sl_ix1 on db2user.stats_log(eventtype, event1);
   create index sl_ix2 on db2user.stats_log(objtype, event1);
   create index sl_ix3 on db2user.stats_log(objname);

   runstats on table db2user.stats_log
     with distribution and sampled detailed indexes all;
   ```

Collecting catalog statistics

Use the **RUNSTATS** utility to collect catalog statistics on tables, indexes, and statistical views. The query optimizer uses this information to choose the best access plans for queries.

About this task

For privileges and authorities that are required to use this utility, see the description of the **RUNSTATS** command.

Procedure

To collect catalog statistics:

1. Connect to the database that contains the tables, indexes, or statistical views for which you want to collect statistical information.
2. Collect statistics for queries that run against the tables, indexes, or statistical views by using one of the following methods:
   - From the DB2 command line, execute the **RUNSTATS** command with appropriate options. These options enable you to tailor the statistics that are collected for queries that run against the tables, indexes, or statistical views.
   - From IBM Data Studio, open the task assistant for the **RUNSTATS** command.
3. When the runstats operation completes, issue a COMMIT statement to release locks.
4. Rebind any packages that access the tables, indexes, or statistical views for which you have updated statistical information.

Results

Note:

1. The **RUNSTATS** command does not support the use of nicknames. If queries access a federated database, use **RUNSTATS** to update statistics for tables in all databases, then drop and recreate the nicknames that access remote tables to make the new statistics available to the optimizer.
2. When you collect statistics for a table in a partitioned database environment, **RUNSTATS** only operates on the database partition from which the utility is executed. The results from this database partition are extrapolated to the other database partitions. If this database partition does not contain a required portion of the table, the request is sent to the first database partition in the database partition group that contains the required data.

   Statistics for a statistical view are collected on all database partitions containing base tables that are referenced by the view.
3. For DB2 V9.7 Fix Pack 1 and later releases, the following apply to the collection of distribution statistics on a column of type XML:
   - Distribution statistics are collected for each index over XML data specified on an XML column.

- The **RUNSTATS** command must collect both distribution statistics and table statistics to collect distribution statistics for an index over XML data.
- As the default, the **RUNSTATS** command collects a maximum of 250 quantiles for distribution statistics for each index over XML data. The maximum number of quantiles for a column can be specified when executing the **RUNSTATS** command.
- Distribution statistics are collected on indexes over XML data of type VARCHAR, DOUBLE, TIMESTAMP, and DATE. XML distribution statistics are not collected on indexes over XML data of type VARCHAR HASHED.
- Distribution statistics are not collected on partitioned indexes over XML data defined on a partitioned table.

Collecting statistics on a sample of the data

Table statistics are used by the query optimizer to select the best access plan for a query, so it is important that statistics remain current. With the ever-increasing size of databases, efficient statistics collection becomes more challenging.

An effective approach is to collect statistics on a random sample of table and index data. For I/O-bound or processor-bound systems, the performance benefits can be enormous.

The DB2 product enables you to efficiently sample data for statistics collection, potentially improving the performance of the **RUNSTATS** utility by orders of magnitude, while maintaining a high degree of accuracy.

Two methods of sampling are available: row-level sampling and page-level sampling. For a description of these sampling methods, see “Data sampling in queries”.

Performance of page-level sampling is excellent, because only one I/O operation is required for each selected page. With row-level sampling, I/O costs are not reduced, because every table page is retrieved in a full table or index scan. However, row-level sampling provides significant performance improvements, even if the amount of I/O is not reduced, because collecting statistics is processor-intensive.

Row-level table sampling provides a better sample than page-level table sampling in situations where the data values are highly clustered. Compared to page-level table sampling, the row-level table sample set will likely be a better reflection of the data, because it will include *P* percent of the rows from each data page. With page-level table sampling, all the rows of *P* percent of the pages will be in the sample set. If the rows are distributed randomly over the table, the accuracy of row-sampled statistics will be similar to the accuracy of page-sampled statistics.

Each table sample is randomly generated across repeated invocations of the **RUNSTATS** command, unless the **REPEATABLE** parameter is used, in which case the previous table sample is regenerated. This option can be useful in cases where consistent statistics are required for tables whose data remains constant.

REPEATABLE does not apply to the index sampling (**INDEXSAMPLE**) - there is no similar functionality.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for collecting statistics. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Collecting index statistics

Collect index statistics to help the optimizer decide whether a specific index should be used to resolve a query.

About this task

The following example is based on a database named SALES that contains a CUSTOMERS table with indexes CUSTIDX1 and CUSTIDX2.

For privileges and authorities that are required to use the **RUNSTATS** utility, see the description of the **RUNSTATS** command.

Procedure

To collect detailed statistics for an index:

1. Connect to the SALES database.
2. Execute one of the following commands from the DB2 command line, depending on your requirements:
   - To collect detailed statistics on both CUSTIDX1 and CUSTIDX2:
     ```
     runstats on table sales.customers and detailed indexes all
     ```
   - To collect detailed statistics on both indexes, but with sampling instead of detailed calculations on each index entry:
     ```
     runstats on table sales.customers and sampled detailed indexes all
     ```

     The **SAMPLED DETAILED** parameter requires 2 MB of the statistics heap. Allocate an additional 488 4-KB pages to the **stat_heap_sz** database configuration parameter setting for this memory requirement. If the heap is too small, the **RUNSTATS** utility returns an error before it attempts to collect statistics.
   - To collect detailed statistics on sampled indexes, as well as distribution statistics for the table so that index and table statistics are consistent:
     ```
     runstats on table sales.customers
       with distribution on key columns
         and sampled detailed indexes all
     ```

Collecting distribution statistics for specific columns

For efficient **RUNSTATS** operations and subsequent query-plan analysis, collect distribution statistics on only those columns that queries reference in WHERE, GROUP BY, and similar clauses. You can also collect cardinality statistics on combined groups of columns. The optimizer uses such information to detect column correlation when it estimates selectivity for queries that reference the columns in a group.

About this task

The following example is based on a database named SALES that contains a CUSTOMERS table with indexes CUSTIDX1 and CUSTIDX2.

For privileges and authorities that are required to use the **RUNSTATS** utility, see the description of the **RUNSTATS** command.

When you collect statistics for a table in a partitioned database environment, **RUNSTATS** operates only on the database partition from which the utility is executed. The results from this database partition are extrapolated to the other database partitions. If this database partition does not contain a required portion of the table, the request is sent to the first database partition in the database partition group that contains the required data.

Procedure

To collect statistics on specific columns:

1. Connect to the SALES database.
2. Execute one of the following commands from the DB2 command line, depending on your requirements:
   - To collect distribution statistics on columns ZIP and YTDTOTAL:
     ```
     runstats on table sales.customers
       with distribution on columns (zip, ytdtotal)
     ```
   - To collect distribution statistics on the same columns, but with different distribution options:
     ```
     runstats on table sales.customers
       with distribution on columns (
         zip, ytdtotal num_freqvalues 50 num_quantiles 75)
     ```
   - To collect distribution statistics on the columns that are indexed in CUSTIDX1 and CUSTIDX2:
     ```
     runstats on table sales.customer
       on key columns
     ```
   - To collect statistics for columns ZIP and YTDTOTAL and a column group that includes REGION and TERRITORY:
     ```
     runstats on table sales.customers
       on columns (zip, (region, territory), ytdtotal)
     ```
   - Suppose that statistics for non-XML columns were collected previously using the **LOAD** command with the **STATISTICS** parameter. To collect statistics for the XML column MISCINFO:
     ```
     runstats on table sales.customers
       on columns (miscinfo)
     ```
   - To collect statistics for the non-XML columns only:
     ```
     runstats on table sales.customers
       excluding xml columns
     ```

     The **EXCLUDING XML COLUMNS** clause takes precedence over all other clauses that specify XML columns.
   - For DB2 V9.7 Fix Pack 1 and later releases, the following command collects distribution statistics using a maximum of 50 quantiles for the XML column MISCINFO. A default of 20 quantiles is used for all other columns in the table:
     ```
     runstats on table sales.customers
        with distribution on columns ( miscinfo num_quantiles 50 )
        default num_quantiles 20
     ```

     Note: The following are required for distribution statistics to be collected on the XML column MISCINFO:
     - Both table and distribution statistics must be collected.

– An index over XML data must be defined on the column, and the data type specified for the index must be VARCHAR, DOUBLE, TIMESTAMP, or DATE.

Monitoring the progress of RUNSTATS operations

You can use the **LIST UTILITIES** command or the **db2pd** command to monitor the progress of **RUNSTATS** operations on a database.

Procedure

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

```
   list utilities show detail
```

or issue the **db2pd** command and specify the **-runstats** parameter:

```
db2pd -runstats
```

Results

The following is an example of the output for monitoring the performance of a **RUNSTATS** operation using the **LIST UTILITIES** command:

```
ID                               = 7
Type                             = RUNSTATS
Database Name                    = SAMPLE
Partition Number                 = 0
Description                      = YIWEIANG.EMPLOYEE
Start Time                       = 08/04/2011 12:39:35.155398
State                            = Executing
Invocation Type                  = User
Throttling:
   Priority                      = Unthrottled
```

The following is an example of the output for monitoring the performance of a **RUNSTATS** operation using the **db2pd** command:

```
db2pd -runstats

Table Runstats Information:

Retrieval Time: 08/13/2009 20:38:20
TbspaceID: 2        TableID: 4
Schema: SCHEMA    TableName: TABLE
Status: Completed     Access: Allow write
Sampling: No          Sampling Rate: -
Start Time: 08/13/2009 20:38:16   End Time: 08/13/2009 20:38:17
Total Duration: 00:00:01
Cur Count: 0                      Max Count: 0

Index Runstats Information:

Retrieval Time: 08/13/2009 20:38:20
TbspaceID: 2        TableID: 4
Schema: SCHEMA    TableName: TABLE
Status: Completed     Access: Allow write
Start Time: 08/13/2009 20:38:17   End Time: 08/13/2009 20:38:18
Total Duration: 00:00:01
Prev Index Duration [1]: 00:00:01
Prev Index Duration [2]:  -
Prev Index Duration [3]:  -
Cur Index Start: 08/13/2009 20:38:18
Cur Index: 2            Max Index: 2            Index ID: 2
Cur Count: 0                      Max Count: 0
```

Minimizing RUNSTATS impact

There are several approaches available to improve **RUNSTATS** performance.

To minimize the performance impact of this utility:

- Limit the columns for which statistics are collected by using the COLUMNS clause. Many columns are never referenced by predicates in the query workload, so they do not require statistics.
- Limit the columns for which distribution statistics are collected if the data tends to be uniformly distributed. Collecting distribution statistics requires more CPU and memory than collecting basic column statistics. However, determining whether the values for a column are uniformly distributed requires either having existing statistics or querying the data. This approach also assumes that the data remains uniformly distributed as the table is modified.
- Limit the number of pages and rows processed by using page- or row-level table sampling (by specifying the **TABLESAMPLE SYSTEM** or **TABLESAMPLE BERNOULLI** clause) and by using page- or row-level index sampling (by specifying **INDEXSAMPLE SYSTEM** or **INDEXSAMPLE BERNOULLI** clause). Start with a 10% page-level sample, by specifying `TABLESAMPLE SYSTEM(10)` and `INDEXSAMPLE SYSTEM(10)`. Check the accuracy of the statistics and whether system performance has degraded due to changes in access plan. If it has degraded, try a 10% row-level sample instead, by specifying `TABLESAMPLE BERNOULLI(10)`. Likewise, experiment with the **INDEXSAMPLE** parameter to get the right rate for index sampling. If the accuracy of the statistics is insufficient, increase the sampling amount. When using **RUNSTATS** page- or row-level sampling, use the same sampling rate for tables that are joined. This is important to ensure that the join column statistics have the same level of accuracy.
- Collect index statistics during index creation by specifying the COLLECT STATISTICS option on the CREATE INDEX statement. This approach is faster than performing a separate **RUNSTATS** operation after the index is created. It also ensures that the new index has statistics generated immediately after creation, to allow the optimizer to accurately estimate the cost of using the index.
- Collect statistics when executing the **LOAD** command with the **REPLACE** option. This approach is faster than performing a separate **RUNSTATS** operation after the load operation completes. It also ensures that the table has the most current statistics immediately after the data is loaded, to allow the optimizer to accurately estimate the cost of using the table.

In a partitioned database environment, the **RUNSTATS** utility collects statistics from a single database partition. If the **RUNSTATS** command is issued on a database partition on which the table resides, statistics are collected there. If not, statistics are collected on the first database partition in the database partition group for the table. For consistent statistics, ensure that statistics for joined tables are collected from the same database partition.

Recompiling a query after configuration changes

To observe the effect of configuration changes that affect query optimization, it might be necessary to cause the query optimizer to recompile the statements that are cached.

Procedure

You can cause the query optimizer to recompile a statement by performing any of the following actions:

- Invalidating the cached dynamic statements for specific tables using the **RUNSTATS** command:

  ```
  RUNSTATS ON TABLE <tableschema>.<tablename>
     WITH DISTRIBUTION AND SAMPLED DETAILED INDEXES ALL
  ```

  Note: This will refresh the table statistics and subsequent query compilations will use the new statistics as well as the new configuration settings.
- Removing all cached dynamic SQL statements currently in the package cache:

  ```
  FLUSH PACKAGE CACHE DYNAMIC
  ```

Avoiding manual updates to the catalog statistics

The DB2 data server supports manually updating catalog statistics by issuing UPDATE statements against views in the SYSSTAT schema.

This feature can be useful when mimicking a production database on a test system in order to examine query access plans. The **db2look** utility is very helpful for capturing the DDL and UPDATE statements against views in the SYSSTAT schema for playback on another system.

Avoid influencing the query optimizer by manually providing incorrect statistics to force a particular query access plan. Although this practice might result in improved performance for some queries, it can result in performance degradation for others. Consider other tuning options (such as using optimization guidelines and profiles) before resorting to this approach. If this approach does become necessary, be sure to record the original statistics in case they need to be restored.

Chapter 41. Binding embedded SQL packages to a database

Binding is the process of creating a package from a bind file and storing it in a database.

Application, bind file, and package relationships

Database applications use packages for some of the same reasons that applications are compiled: improved performance and compactness. By precompiling an SQL statement, the statement is compiled into the package when the application is built, instead of at run time. Each statement is parsed, and a more efficiently interpreted operand string is stored in the package. At run time, the code generated by the precompiler calls run-time services database manager APIs with any variable information required for input or output data, and the information stored in the package is executed.

The advantages of precompilation apply only to static SQL statements. SQL statements that are executed dynamically (using PREPARE and EXECUTE or EXECUTE IMMEDIATE) are not precompiled; therefore, they must go through the entire set of processing steps at run time.

With the DB2 bind file description (**db2bfd**) utility, you can easily display the contents of a bind file to examine and verify the SQL statements within it. You can also display the precompile options used to create the bind file using the DB2 bind file description (**db2bfd**) utility. This can be useful in problem determination related to the bind file for your application.

You can set the STATICASDYNAMIC string on the **GENERIC** parameter of the **BIND** command to "yes" to instruct the DB2 database manager to store all statements in the catalogs and mark them as incremental bind. At run time, when the package is first loaded, the database manager uses the current session environment (rather than the package) to set up the section entries and other entities (text is populated and the package cache is accessed). Thereafter, the statements in the bound file behave the same as they would if you were using dynamic SQL. For example, sections will be implicitly recompiled for Database Definition Language invalidations, special register updates, and so on. The DB2 database manager provides this feature to facilitate the migration of embedded SQL C applications from other database systems.

Effect of DYNAMICRULES bind option on dynamic SQL

The **PRECOMPILE** command and **BIND** command parameter **DYNAMICRULES** determines which rules apply to dynamic SQL at run time.

In particular, the **DYNAMICRULES** parameter determines what values apply at run time for the following dynamic SQL attributes:

- The authorization ID that is used during authorization checking.
- The qualifier that is used for qualification of unqualified objects.
- Whether the package can be used to dynamically prepare the following statements: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY, and SET EVENT MONITOR STATE statements.

In addition to the **DYNAMICRULES** value, the runtime environment of a package controls how dynamic SQL statements behave at run time. The two possible runtime environments are:

- The package runs as part of a stand-alone program
- The package runs within a routine context

The combination of the **DYNAMICRULES** value and the runtime environment determine the values for the dynamic SQL attributes. That set of attribute values is called the dynamic SQL statement behavior. The four behaviors are:

Run behavior
DB2 for Linux, UNIX, and Windows uses the authorization ID of the user (the ID that initially connected to the DB2 database) executing the package as the value to be used for authorization checking of dynamic SQL statements and for the initial value used for implicit qualification of unqualified object references within dynamic SQL statements.

Bind behavior
At run time, DB2 for Linux, UNIX, and Windows uses all the rules that apply to static SQL for authorization and qualification. That is, take the authorization ID of the package owner as the value to be used for authorization checking of dynamic SQL statements and the package default qualifier for implicit qualification of unqualified object references within dynamic SQL statements.

Define behavior
Define behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with **DYNAMICRULES** DEFINEBIND or **DYNAMICRULES** DEFINERUN. DB2 for Linux, UNIX, and Windows uses the authorization ID of the routine definer (not the routine's package binder) as the value to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

Invoke behavior
Invoke behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with **DYNAMICRULES** INVOKEBIND or **DYNAMICRULES** INVOKERUN. DB2 for Linux, UNIX, and Windows uses the current statement authorization ID in effect when the routine is invoked as the value to be used for authorization checking of dynamic SQL and for implicit qualification of unqualified object references within dynamic SQL statements within that routine. This is summarized by the following table:

Invoking Environment	ID Used
Any static SQL	Implicit or explicit value of the **OWNER** of the package the SQL invoking the routine came from.
Used in definition of view or trigger	Definer of the view or trigger.
Dynamic SQL from a run behavior package	ID used to make the initial connection to the DB2 database.
Dynamic SQL from a define behavior package	Definer of the routine that uses the package that the SQL invoking the routine came from.

Invoking Environment	ID Used
Dynamic SQL from an invoke behavior package	Current authorization ID invoking the routine.

The following table shows the combination of the **DYNAMICRULES** value and the runtime environment that yields each dynamic SQL behavior.

Table 120. How DYNAMICRULES and the Runtime Environment Determine Dynamic SQL Statement Behavior

DYNAMICRULES Value	Behavior of Dynamic SQL Statements in a Standalone Program Environment	Behavior of Dynamic SQL Statements in a Routine Environment
BIND	Bind behavior	Bind behavior
RUN	Run behavior	Run behavior
DEFINEBIND	Bind behavior	Define behavior
DEFINERUN	Run behavior	Define behavior
INVOKEBIND	Bind behavior	Invoke behavior
INVOKERUN	Run behavior	Invoke behavior

The following table shows the dynamic SQL attribute values for each type of dynamic SQL behavior.

Table 121. Definitions of Dynamic SQL Statement Behaviors

Dynamic SQL Attribute	Setting for Dynamic SQL Attributes: Bind Behavior	Setting for Dynamic SQL Attributes: Run Behavior	Setting for Dynamic SQL Attributes: Define Behavior	Setting for Dynamic SQL Attributes: Invoke Behavior
Authorization ID	The implicit or explicit value of the **BIND OWNER** command parameter	ID of User Executing Package	Routine definer (not the routine's package owner)	Current statement authorization ID when routine is invoked.
Default qualifier for unqualified objects	The implicit or explicit value of the **BIND QUALIFIER** command parameter	CURRENT SCHEMA Special Register	Routine definer (not the routine's package owner)	Current statement authorization ID when routine is invoked.
Can execute GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY, and SET EVENT MONITOR STATE	No	Yes	No	No

Bind considerations

If your application uses a code page that differs from the database code page, you must ensure that the code page used by the application is compatible with the database code page during the bind process.

If your application issues calls to any of the database manager utility APIs, such as **IMPORT** or **EXPORT**, you must bind the supplied utility bind files to the database.

You can use bind options to control certain operations that occur during binding, as in the following examples:

- The **QUERYOPT** bind parameter takes advantage of a specific optimization class when binding.
- The **EXPLSNAP** bind parameter stores Explain Snapshot information for eligible SQL statements in the Explain tables.
- The **FUNCPATH** bind parameter properly resolves user-defined distinct types and user-defined functions in static SQL.

If the bind process starts but never returns, it might be that other applications connected to the database hold locks that you require. In this case, ensure that no applications are connected to the database. If they are, disconnect all applications on the server and the bind process will continue.

If your application will access a server using DB2 Connect, you can use the **BIND** command parameters available for that server.

Bind files are not compatible with earlier versions of DB2 for Linux, UNIX, and Windows. In mixed-level environments, DB2 for Linux, UNIX, and Windows can only use the functions available to the lowest level of the database environment. For example, if a version 8 client connects to a version 7.2 server, the client will only be able to use version 7.2 functions. As bind files express the functionality of the database, they are subject to the mixed-level restriction.

If you need to rebind higher-level bind files on lower-level systems, you can:

- Use a lower level IBM data server client to connect to the higher-level server and create bind files which can be shipped and bound to the lower-level DB2 for Linux, UNIX, and Windows environment.
- Use a higher-level IBM data server client in the lower-level production environment to bind the higher-level bind files that were created in the test environment. The higher-level client passes only the options that apply to the lower-level server.

Performance improvements when using REOPT option of the BIND command

The bind option **REOPT** can significantly improve the embedded SQL application performance.

Effects of REOPT on static SQL

The bind option **REOPT** can make static SQL statements containing host variables, global variables, or special registers behave like incremental-bind statements. This means that these statements get compiled at the time of EXECUTE or OPEN instead of at bind time. During this compilation, the access plan is chosen, based on the real values of these variables.

With **REOPT ONCE**, the access plan is cached after the first OPEN or EXECUTE request and is used for subsequent execution of this statement. With **REOPT ALWAYS**, the access plan is regenerated for every OPEN and EXECUTE request, and the current set of host variable, parameter marker, global variable, and special register values is used to create this plan.

Effects of REOPT on dynamic SQL

When you specify the option **REOPT ALWAYS**, the database manager postpones preparing any statement containing host variables, parameter markers, global variables, or special registers until it encounters an OPEN or EXECUTE statement; that is, when the values for these variables become known. At this time, the access plan is generated using these values. Subsequent OPEN or EXECUTE requests for the same statement will recompile the statement, reoptimize the query plan using the current set of values for the variables, and execute the newly generated query plan. When **REOPT ALWAYS** is specified, statement concentrator is disabled.

The option **REOPT ONCE** has a similar effect, with the exception that the plan is only optimized once using the values of the host variables, parameter markers, global variables, and special registers. This plan is cached and will be used by subsequent requests.

Binding applications with the BIND command

Binding is the process that creates the package the database manager needs to access the database when the application is executed.

By default the **PRECOMPILE** command creates a package. Binding is done implicitly at precompile time unless the **BINDFILE** command parameter is specified. The **PACKAGE** command parameter allows you to specify a package name for the package created at precompile time.

A typical example of using the **BIND** command follows. To bind a bind file named `filename.bnd` to the database, you can issue the following command:

```
BIND filename.bnd
```

One package is created for each separately precompiled source code module. If an application has five source files, of which three require precompilation, three packages or bind files are created. By default, each package is given a name that is the same as the name of the source module from which the `.bnd` file originated, but truncated to 8 characters. To explicitly specify a different package name, you must use the **PACKAGE USING** parameter on the **PREP** command. The version of a package is given by the **VERSION** precompile parameter and defaults to the empty string. If the name and schema of this newly created package is the same as a package that currently exists in the target database, but the version identifier differs, a new package is created and the previous package still remains. However if a package exists that matches the name, schema and the version of the package being bound, then that package is dropped and replaced with the new package being bound (specifying **ACTION ADD** on the bind would prevent that and an error (SQL0719) would be returned instead).

Rebinding existing packages with the REBIND command

Rebinding is the process of recreating a package for an application program that was previously bound. You must rebind packages if they were marked invalid or inoperative or if the database statistics changed since the last binding.

In some situations, however, you might want to rebind packages that are valid. For example, you might want to take advantage of a newly created index, or use updated statistics after executing the **RUNSTATS** command.

Packages can be dependent on certain types of database objects such as tables, views, aliases, indexes, triggers, referential constraints, and table check constraints. If a package is dependent on a database object (such as a table, view, trigger, and so on), and that object is dropped, the package is placed into an invalid state. If the object that is dropped is a UDF, the package is placed into an inoperative state.

When the package is marked inoperative, the next use of a statement in this package causes an implicit rebind of the package using non-conservative binding semantics in order to be able to resolve to SQL objects considering the latest changes in the database schema that caused that package to become inoperative.

For static DML in packages, the packages can rebind implicitly, or by explicitly issuing the **REBIND** command (or corresponding API), or the **BIND** command (or corresponding API). The implicit rebind is performed with conservative binding semantics if the package is marked invalid, but uses non-conservative binding semantics when the package is marked inoperative.

You must use the **BIND** command to rebind a package for a program which was modified to include more, fewer, or changed SQL statements. You must also use the **BIND** command if you need to change any bind options from the values with which the package was originally bound. The **REBIND** command provides the option to resolve with conservative binding semantics (**RESOLVE CONSERVATIVE**) or to resolve by considering new routines, data types, or global variables (**RESOLVE ANY**, which is the default option). The **RESOLVE CONSERVATIVE** option can be used only if the package was not marked inoperative by the database manager (SQLSTATE 51028). You should use **REBIND** whenever your situation does not specifically require the use of **BIND**, as the performance of **REBIND** is significantly better than that of **BIND**.

When multiple versions of the same package name coexist in the catalog, only one version can be rebound at a time.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for rebinding packages. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Binding utilities to the database

When a database is created, the database manager attempts to bind the utilities in db2ubind.lst and in db2cli.lst to the database. These files are stored in the bnd subdirectory of your sqllib directory.

About this task

Binding a utility creates a *package*, which is an object that includes all the information needed to process specific SQL and XQuery statements from a single source file.

Note: If you want to use these utilities from a client, you must bind them explicitly. You must be in the directory where these files reside to create the packages in the sample database. The bind files are found in the bnd subdirectory of the sqllib directory. You must also bind the db2schema.bnd file when you create or upgrade the database from a client. See "DB2 CLI bind files and package names" for details.

Procedure

To bind or rebind the utilities to a database, from the command line, invoke the following commands:

```
connect to sample
bind @db2ubind.lst
```

where *sample* is the name of the database.

Binding applications and utilities (DB2 Connect server)

Application programs developed using embedded SQL must be bound to each database with which they will operate. For information about the binding requirements for the IBM data server package, see the topic about DB2 CLI bind files and package names.

Binding should be performed once per application, for each database. During the bind process, database access plans are stored for each SQL statement that will be executed. These access plans are supplied by application developers and are contained in *bind files* which are created during precompilation. Binding is a process of processing these bind files by an IBM mainframe database server.

Because several of the utilities supplied with DB2 Connect are developed using embedded SQL, they must be bound to an IBM mainframe database server before they can be used with that system. If you do not use the DB2 Connect utilities and interfaces, you do not have to bind them to each of your IBM mainframe database servers. The lists of bind files required by these utilities are contained in the following files:

- `ddcsmvs.lst` for System z
- `ddcsvse.lst` for VSE
- `ddcsvm.lst` for VM
- `ddcs400.lst` for IBM Power Systems™

Binding one of these lists of files to a database will bind each individual utility to that database.

If a DB2 Connect server product is installed, the DB2 Connect utilities must be bound to each IBM mainframe database server before they can be used with that system. Assuming the clients are at the same fix pack level, you need to bind the utilities only once, regardless of the number of client platforms involved.

For example, if you have 10 Windows clients, and 10 AIX clients connecting to DB2 for z/OS via DB2 Connect Enterprise Edition on a Windows server, perform one of the following steps:

- Bind `ddcsmvs.lst` from one of the Windows clients.
- Bind `ddcsmvs.lst` from one of the AIX clients.
- Bind `ddcsmvs.lst` from the DB2 Connect server.

This example assumes that:

- All the clients are at the same service level. If they are not then, in addition, you might need to bind from each client of a particular service level
- The server is at the same service level as the clients. If it is not, then you need to bind from the server as well.

In addition to DB2 Connect utilities, any other applications that use embedded SQL must also be bound to each database that you want them to work with. An application that is not bound will usually produce an SQL0805N error message when executed. You might want to create an additional bind list file for all of your applications that need to be bound.

For each IBM mainframe database server that you are binding to, perform the following steps:

1. Make sure that you have sufficient authority for your IBM mainframe database server management system:

   System z

   The authorizations required are:

   - SYSADM or
   - SYSCTRL or
   - BINDADD *and* CREATE IN COLLECTION NULLID

   Note: The BINDADD and the CREATE IN COLLECTION NULLID privileges provide sufficient authority **only** when the packages do not already exist. For example, if you are creating them for the first time.

   If the packages already exist, and you are binding them again, then the authority required to complete the task(s) depends on who did the original bind.

   A) If you did the original bind and you are doing the bind again, then having any of the previously listed authorities will allow you to complete the bind.

   B) If your original bind was done by someone else and you are doing the second bind, then you will require either the SYSADM or the SYSCTRL authorities to complete the bind. Having just the BINDADD and the CREATE IN COLLECTION NULLID authorities will not allow you to complete the bind. It is still possible to create a package if you do not have either SYSADM or SYSCTRL privileges. In this situation you would need the BIND privilege on each of the existing packages that you intend to replace.

   VSE or VM

   The authorization required is DBA authority. If you want to use the GRANT option on the bind command (to avoid granting access to each DB2 Connect package individually), the NULLID user ID must have the authority to grant authority to other users on the following tables:

   - system.syscatalog
   - system.syscolumns
   - system.sysindexes
   - system.systabauth
   - system.syskeycols
   - system.syssynonyms
   - system.syskeys
   - system.syscolauth
   - system.sysuserauth

   On the VSE or VM system, you can issue:

```
          grant select on table to nullid with grant option
```

IBM Power Systems
*CHANGE authority or higher on the NULLID collection.

2. Issue commands similar to the following commands:

   ```
   db2 connect to DBALIAS user USERID using PASSWORD
   db2 bind path@ddcsmvs.lst blocking all
         sqlerror continue messages ddcsmvs.msg grant public
   db2 connect reset
   ```

   Where *DBALIAS*, *USERID*, and *PASSWORD* apply to the IBM mainframe database server, ddcsmvs.lst is the bind list file for z/OS, and *path* represents the location of the bind list file.

   For example *drive*:\sqllib\bnd\ applies to all Windows operating systems, and *INSTHOME*/sqllib/bnd/ applies to all Linux and UNIX operating systems, where *drive* represents the logical drive where DB2 Connect was installed and *INSTHOME* represents the home directory of the DB2 Connect instance.

   You can use the grant option of the **bind** command to grant EXECUTE privilege to PUBLIC or to a specified user name or group ID. If you do not use the grant option of the **bind** command, you must GRANT EXECUTE (RUN) individually.

   To find out the package names for the bind files, enter the following command:

   ```
   ddcspkgn @bindfile.lst
   ```

   For example:

   ```
   ddcspkgn @ddcsmvs.lst
   ```

   might yield the following output:

   ```
   Bind File                      Package Name
   ------------------------------ ------------------------------
   f:\sqllib\bnd\db2ajgrt.bnd     SQLAB6D3
   ```

   To determine these values for DB2 Connect execute the **ddcspkgn** utility, for example:

   ```
   ddcspkgn @ddcsmvs.lst
   ```

   Optionally, this utility can be used to determine the package name of individual bind files, for example:

   ```
   ddcspkgn bindfile.bnd
   ```

   Note:

   a. Using the bind option **sqlerror continue** is required; however, this option is automatically specified for you when you bind applications using the DB2 tools or the Command Line Processor (CLP). Specifying this option turns bind errors into warnings, so that binding a file containing errors can still result in the creation of a package. In turn, this allows one bind file to be used against multiple servers even when a particular server implementation might flag the SQL syntax of another to be invalid. For this reason, binding any of the list files ddcsxxx.lst against any particular IBM mainframe database server should be expected to produce some warnings.
   b. If you are connecting to a DB2 database through DB2 Connect, use the bind list db2ubind.lst and do not specify **sqlerror continue**, which is only valid when connecting to a IBM mainframe database server. Also, to connect to a DB2 database, it is recommended that you use the DB2 clients provided with DB2 and not DB2 Connect.

3. Use similar statements to bind each application or list of applications.

4. If you have remote clients from a previous release of DB2, you might need to bind the utilities on these clients to DB2 Connect.

Chapter 42. Design Advisor

The DB2 Design Advisor is a tool that can help you significantly improve your workload performance. The task of selecting which indexes, materialized query tables (MQTs), clustering dimensions, or database partitions to create for a complex workload can be daunting. The Design Advisor identifies all of the objects that are needed to improve the performance of your workload.

Given a set of SQL statements in a workload, the Design Advisor generates recommendations for:

- New indexes
- New clustering indexes
- New MQTs
- Conversion to multidimensional clustering (MDC) tables
- The redistribution of tables

The Design Advisor can implement some or all of these recommendations immediately, or you can schedule them to run at a later time.

Use the **db2advis** command to launch the Design Advisor utility.

The Design Advisor can help simplify the following tasks:

Planning for and setting up a new database

While designing your database, use the Design Advisor to generate design alternatives in a test environment for indexing, MQTs, MDC tables, or database partitioning.

In partitioned database environments, you can use the Design Advisor to:

- Determine an appropriate database partitioning strategy before loading data into a database
- Assist in upgrading from a single-partition database to a multi-partition database
- Assist in migrating from another database product to a multi-partition DB2 database

Workload performance tuning

After your database is set up, you can use the Design Advisor to:

- Improve the performance of a particular statement or workload
- Improve general database performance, using the performance of a sample workload as a gauge
- Improve the performance of the most frequently executed queries, as identified, for example, by the IBM InfoSphere Optim Performance Manager
- Determine how to optimize the performance of a new query
- Respond to Data Studio Health Monitor recommendations regarding shared memory utility or sort heap problems with a sort-intensive workload
- Find objects that are not used in a workload

IBM InfoSphere Optim Query Workload Tuner provides tools for improving the performance of single SQL statements and the performance of groups of SQL statements, which are called query workloads. For more information about this product, see the product overview page at http://www.ibm.com/software/data/optim/query-workload-tuner-db2-luw/index.html. In Version 3.1.1 or later, you can also use the Workload Design Advisor to perform many operations that were available in the DB2 Design Advisor wizard. For more information see the documentation for the Workload Design Advisor at http://publib.boulder.ibm.com/infocenter/dstudio/v3r1/topic/com.ibm.datatools.qrytune.workloadtunedb2luw.doc/topics/genrecsdsgn.html.

Design Advisor output

Design Advisor output is written to standard output by default, and saved in the ADVISE_* tables:

- The ADVISE_INSTANCE table is updated with one new row each time that the Design Advisor runs:
  - The START_TIME and END_TIME fields show the start and stop times for the utility.
  - The STATUS field contains a value of COMPLETED if the utility ended successfully.
  - The MODE field indicates whether the **-m** parameter was used on the **db2advis** command.
  - The COMPRESSION field indicates the type of compression that was used.
- The USE_TABLE column in the ADVISE_TABLE table contains a value of Y if MQT, MDC table, or database partitioning strategy recommendations were made.

  MQT recommendations can be found in the ADVISE_MQT table; MDC recommendations can be found in the ADVISE_TABLE table; and database partitioning strategy recommendations can be found in the ADVISE_PARTITION table. The RUN_ID column in these tables contains a value that corresponds to the START_TIME value of a row in the ADVISE_INSTANCE table, linking it to the same Design Advisor run.

  When MQT, MDC, or database partitioning recommendations are provided, the relevant ALTER TABLE stored procedure call is placed in the ALTER_COMMAND column of the ADVISE_TABLE table. The ALTER TABLE stored procedure call might not succeed due to restrictions on the table for the ALTOBJ stored procedure.
- The USE_INDEX column in the ADVISE_INDEX table contains a value of Y (index recommended or evaluated) or R (an existing clustering RID index was recommended to be unclustered) if index recommendations were made.
- The COLSTATS column in the ADVISE_MQT table contains column statistics for an MQT. These statistics are contained within an XML structure as follows:

  ```
  <?xml version=\"1.0\" encoding=\"USASCII\"?>
  <colstats>
      <column>
           <name>COLNAME1</name>
           <colcard>1000</colcard>
           <high2key>999</high2key>
           <low2key>2</low2key>
      </column>
      ....

      <column>
           <name>COLNAME100</name>
           <colcard>55000</colcard>
  ```

```
          <high2key>49999</high2key>
          <low2key>100</low2key>
      </column>
  </colstats>
```

You can save Design Advisor recommendations to a file using the **-o** parameter on the **db2advis** command. The saved Design Advisor output consists of the following elements:

- CREATE statements associated with any new indexes, MQTs, MDC tables, or database partitioning strategies
- REFRESH statements for MQTs
- **RUNSTATS** commands for new objects

An example of this output is as follows:

```
--<?xml version="1.0"?>
--<design-advisor>
--<mqt>
--<identifier>
--<name>MQT612152202220000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<statementlist>3</statementlist>
--<benefit>1013562.481682</benefit>
--<overhead>1468328.200000</overhead>
--<diskspace>0.004906</diskspace>
--</mqt>
.....
--<index>
--<identifier>
--<name>IDX612152221400000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<table><identifier>
--<name>PART</name>
--<schema>SAMP </schema>
--</identifier></table>
--<statementlist>22</statementlist>
--<benefit>820160.000000</benefit>
--<overhead>0.000000</overhead>
--<diskspace>9.063500</diskspace>
--</index>
.....
--<statement>
--<statementnum>11</statementnum>
--<statementtext>
--
-- select
-- c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice,
-- sum(l_quantity) from samp.customer, samp.orders,
-- samp.lineitem where o_orderkey in( select
-- l_orderkey from samp.lineitem group by l_orderkey
-- having sum(l_quantity) > 300 ) and c_custkey
-- = o_custkey and o_orderkey = l_orderkey group by
-- c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
-- order by o_totalprice desc, o_orderdate fetch first
-- 100 rows only
--</statementtext>
--<objects>
--<identifier>
--<name>MQT612152202490000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>ORDERS</name>
```

```
--<schema>SAMP </schema>
--</identifier>
--<identifier>
--<name>CUSTOMER</name>
--<schema>SAMP </schema>
--</identifier>
--<identifier>
--<name>IDX612152235020000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>IDX612152235030000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>IDX612152211360000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--</objects>
--<benefit>2091459.000000</benefit>
--<frequency>1</frequency>
--</statement>
```

This XML structure can contain more than one column. The column cardinality (that is, the number of values in each column) is included and, optionally, the HIGH2KEY and LOW2KEY values.

The base table on which an index is defined is also included. Ranking of indexes and MQTs can be done using the benefit value. You can also rank indexes using (benefit - overhead) and MQTs using (benefit - 0.5 * overhead).

Following the list of indexes and MQTs is the list of statements in the workload, including the SQL text, the statement number for the statement, the estimated performance improvement (benefit) from the recommendations, and the list of tables, indexes, and MQTs that were used by the statement. The original spacing in the SQL text is preserved in this output example, but the SQL text is normally split into 80 character commented lines for increased readability.

Existing indexes or MQTs are included in the output if they are being used to execute a workload.

MDC and database partitioning recommendations are not explicitly shown in this XML output example.

After some minor modifications, you can run this output file as a CLP script to create the recommended objects. The modifications that you might want to perform include:

- Combining all of the **RUNSTATS** commands into a single **RUNSTATS** invocation against the new or modified objects
- Providing more usable object names in place of system-generated IDs
- Removing or commenting out any data definition language (DDL) for objects that you do not want to implement immediately

Defining a workload for the Design Advisor

When the Design Advisor analyzes a specific workload, it considers factors such as the type of statements that are included in the workload, the frequency with which a particular statement occurs, and characteristics of your database to generate recommendations that minimize the total cost of running the workload.

About this task

A *workload* is a set of SQL statements that the database manager must process during a period of time. The Design Advisor can be run against:

- A single SQL statement that you enter inline with the **db2advis** command
- A set of dynamic SQL statements that were captured in a DB2 snapshot
- A set of SQL statements that are contained in a workload file

You can create a workload file or modify a previously existing workload file. You can import statements into the file from several sources, including:

- A delimited text file
- An event monitor table
- Explained statements in the EXPLAIN_STATEMENT table
- Recent SQL statements that were captured with a DB2 snapshot
- Workload manager activity tables
- Workload manager event monitor tables by using the **-wlm** option from the command line

After you import the SQL statements into a workload file, you can add, change, modify, or remove statements and modify their frequency.

Procedure

- To run the Design Advisor against dynamic SQL statements:
  1. Reset the database monitor with the following command:

     ```
     db2 reset monitor for database database-name
     ```
  2. Wait for an appropriate amount of time to allow for the execution of dynamic SQL statements against the database.
  3. Invoke the **db2advis** command using the **-g** parameter. If you want to save the dynamic SQL statements in the ADVISE_WORKLOAD table for later reference, use the **-p** parameter as well.
- To run the Design Advisor against a set of SQL statements in a workload file:
  1. Create a workload file manually, separating each SQL statement with a semicolon, or import SQL statements from one or more of the sources listed previously.
  2. Set the frequency of the statements in the workload. Every statement in a workload file is assigned a frequency of 1 by default. The frequency of an SQL statement represents the number of times that the statement occurs within a workload relative to the number of times that other statements occur. For example, a particular SELECT statement might occur 100 times in a workload, whereas another SELECT statement occurs 10 times. To represent the relative frequency of these two statements, you can assign the first SELECT statement a frequency of 10; the second SELECT statement has a frequency of 1. You can manually change the frequency or weight of a particular statement in the workload by inserting the following line after the statement: `- - # SET FREQUENCY` *n*, where *n* is the frequency value that you want to assign to the statement.
  3. Invoke the **db2advis** command using the **-i** parameter followed by the name of the workload file.
- To run the Design Advisor against a workload that is contained in the ADVISE_WORKLOAD table, invoke the **db2advis** command using the **-w** parameter followed by the name of the workload.

Design Advisor limitations and restrictions

There are certain limitations and restrictions associated with Design Advisor recommendations about indexes, materialized query tables (MQTs), multidimensional clustering (MDC) tables, and database partitioning.

Restrictions on index recommendations

- Indexes that are recommended for MQTs are designed to improve workload performance, not MQT refresh performance.
- A clustering RID index is recommended only for MDC tables. The Design Advisor will include clustering RID indexes as an option rather than create an MDC structure for the table.
- The Version 9.7 Design Advisor does not recommend partitioned indexes on a partitioned table. All indexes are recommended with an explicit NOT PARTITIONED clause.

Restrictions on MQT recommendations

- The Design Advisor will not recommend incremental MQTs. If you want to create incremental MQTs, you can convert REFRESH IMMEDIATE MQTs into incremental MQTs with your choice of staging tables.
- Indexes that are recommended for MQTs are designed to improve workload performance, not MQT refresh performance.
- If update, insert, or delete operations are not included in the workload, the performance impact of updating a recommended REFRESH IMMEDIATE MQT is not considered.
- It is recommended that REFRESH IMMEDIATE MQTs have unique indexes created on the implied unique key, which is composed of the columns in the GROUP BY clause of the MQT query definition.

Restrictions on MDC recommendations

- An existing table must be populated with sufficient data before the Design Advisor considers MDC for the table. A minimum of twenty to thirty megabytes of data is recommended. Tables that are smaller than 12 extents are excluded from consideration.
- MDC recommendations for new MQTs will not be considered unless the sampling option, -r, is used with the **db2advis** command.
- The Design Advisor does not make MDC recommendations for typed, temporary, or federated tables.
- Sufficient storage space (approximately 1% of the table data for large tables) must be available for the sampling data that is used during the execution of the **db2advis** command.
- Tables that have not had statistics collected are excluded from consideration.
- The Design Advisor does not make recommendations for multicolumn dimensions.

Restrictions on database partitioning recommendations

The Design Advisor can recommend database partitioning only for DB2 Enterprise Server Edition.

Additional restrictions

Temporary simulation catalog tables are created when the Design Advisor runs. An incomplete run can result in some of these tables not being dropped. In this situation, you can use the Design Advisor to drop these tables by restarting the utility. To remove the simulation catalog tables, specify both the -f option and the -n option (for -n, specifying the same user name that was used for the incomplete execution). If you do not specify the -f option, the Design Advisor will only generate the DROP statements that are required to remove the tables; it will not actually remove them.

Note: As of Version 9.5, the -f option is the default. This means that if you run **`db2advis`** with the MQT selection, the database manager automatically drops all local simulation catalog tables using the same user ID as the schema name.

You should create a separate table space on the catalog database partition for storing these simulated catalog tables, and set the DROPPED TABLE RECOVERY option on the CREATE or ALTER TABLESPACE statement to OFF. This enables easier cleanup and faster Design Advisor execution.

Part 6. High availability

The availability of a database solution is a measure of how successful user applications are at performing their required database tasks.

If user applications cannot connect to the database, or if their transactions fail because of errors or time out because of load on the system, the database solution is not very available. If user applications are successfully connecting to the database and performing their work, the database solution is highly available.

Designing a highly available database solution, or increasing the availability of an existing solution requires an understanding of the needs of the applications accessing the database. To get the greatest benefit from the expense of additional storage space, faster processors, or more software licenses, focus on making your database solution as available as required to the most important applications for your business at the time when those applications need it most.

Unplanned outages

Unexpected system failures that could affect the availability of your database solution to users include: power interruption; network outage; hardware failure; operating system or other software errors; and complete system failure in the event of a disaster. If such a failure occurs at a time when users expect to be able to do work with the database, a highly available database solution must do the following:

- Shield user applications from the failure, so the user applications are not aware of the failure. For example, DB2 Data Server can reroute database client connections to alternate database servers if a database server fails.
- Respond to the failure to contain its effect. For example, if a failure occurs on one machine in a cluster, the cluster manager can remove that machine from the cluster so that no further transactions are routed to be processed on the failed machine.
- Recover from the failure to return the system to normal operations. For example, if standby database takes over database operations for a failed primary database, the failed database might restart, recover, and take over once again as the primary database.

These three tasks must be accomplished with a minimum effect on the availability of the solution to user applications.

Planned outage

In a highly available database solution, the impact of maintenance activities on the availability of the database to user applications must be minimized as well.

For example, if the database solution serves a traditional store front that is open for business between the hours of 9am to 5pm, then maintenance activities can occur offline, outside of those business hours without affecting the availability of the database for user applications. If the database solution serves an online banking business that is expected to be available for customers to access through the Internet 24 hours per day, then maintenance activities must be run online, or scheduled for off-peak activity periods to have minimal impact on the availability of the database to the customers.

When you are making business decisions and design choices about the availability of your database solution, you must weigh the following two factors:

- The cost to your business of the database being unavailable to customers
- The cost of implementing a certain degree of availability

For example, consider an Internet-based business that makes a certain amount of revenue, X, every hour the database solution is serving customers. A high availability strategy that saves 10 hours of downtime per year will earn the business 10X extra revenue per year. If the cost of implementing this high availability strategy is less than the expected extra revenue, it would be worth implementing.

Chapter 43. Data recovery

Data recovery is the rebuilding of a database or table space after a problem such as media or storage failure, power interruption, or application failure. If you back up your database, or individual table spaces, you can rebuild them if they become damaged or corrupted in some way.

There are four types of recovery:

- Crash recovery protects a database from being left in an inconsistent, or unusable, state when transactions (also called units of work) are interrupted unexpectedly.
- Disaster recovery consist of the process to restore a database in the event of a fire, earthquake, vandalism, or other catastrophic events.
- Version recovery is the restoration of a previous version of the database, using an image that was created during a backup operation.
- Rollforward recovery can be used to reapply changes that were made by transactions that were committed after a backup was made.

The DB2 database manager starts crash recovery automatically to attempt to recover a database after a power interruption. You can use version recovery or rollforward recovery to recover a damaged database.

Crash recovery

Transactions (or units of work) against a database can be interrupted unexpectedly. If a failure occurs before all of the changes that are part of the unit of work are completed, committed, and written to disk, the database is left in an inconsistent and unusable state.

Crash recovery is the process by which the database is moved back to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred (Figure 52 on page 724). When a database is in a consistent and usable state, it has attained what is known as a *point of consistency*.

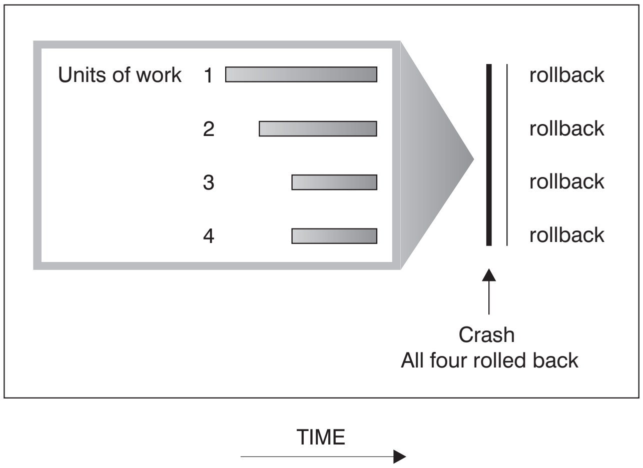


Figure 52. Rolling back units of work (crash recovery)

If you are using the IBM DB2 pureScale Feature, there are two specific types of crash recovery to be aware of: *member crash recovery* and *group crash recovery*. Member crash recovery is the process of recovering a portion of a database using a single member's log stream after a member failure. Member crash recovery, which is usually initiated automatically as a part of a member restart, is an online operation-meaning that other members can still access the database. Multiple members can be undergoing member crash recovery at the same time. Group crash recovery is the process of recovering a database using multiple members' log streams after a failure that causes no viable cluster caching facility to remain in the cluster. Group crash recovery is also usually initiated automatically (as a part of a group restart) and the database is inaccessible while it is in progress, as with DB2 crash recovery operations outside of a DB2 pureScale environment.

If the database or the database manager fails, the database can be left in an inconsistent state. The contents of the database might include changes made by transactions that were incomplete at the time of failure. The database might also be missing changes that were made by transactions that completed before the failure but which were not yet flushed to disk. A crash recovery operation must be performed in order to roll back the partially completed transactions and to write to disk the changes of completed transactions that were previously made only in memory.

Conditions that can necessitate a crash recovery include:

- A power failure on the machine, causing the database manager and the database partitions on it to go down
- A hardware failure such as memory, disk, CPU, or network failure.
- A serious operating system error that causes the DB2 instance to end abnormally

If you want crash recovery to be performed automatically by the database manager, enable the automatic restart (**autorestart**) database configuration parameter by setting it to ON. (This is the default value.) If you do not want automatic restart behavior, set the **autorestart** database configuration parameter to OFF. As a result, you must issue the **RESTART DATABASE** command when a database failure occurs. If the database I/O was suspended before the crash occurred, you must specify the **WRITE RESUME** option of the **RESTART DATABASE** command in order for the crash recovery to continue. The administration notification log records when the database restart operation begins.

If crash recovery occurs on a database that is enabled for rollforward recovery (that is, the **logarchmeth1** configuration parameter is not set to OFF), and an error occurs during crash recovery that is attributable to an individual table space, that table space is taken offline, and cannot be accessed until it is repaired. Crash recovery continues on other table spaces. At the completion of crash recovery, the other table spaces in the database are accessible, and connections to the database can be established. However, if the table space that is taken offline is the table space that contains the system catalogs, it must be repaired before any connections are permitted. This behavior does not apply to DB2 pureScale environments. If an error occurs during member crash recovery or group crash recovery, the crash recovery operation fails.

Recovering damaged table spaces

A damaged table space has one or more containers that cannot be accessed. This is often caused by media problems that are either permanent (for example, a bad disk), or temporary (for example, an offline disk, or an unmounted file system).

If the damaged table space is the system catalog table space, the database cannot be restarted. If the container problems cannot be fixed leaving the original data intact, the only available options are:

- To restore the database
- To restore the catalog table space.

  Note:

  1. Table space restore is only valid for recoverable databases, because the database must be rolled forward.
  2. If you restore the catalog table space, you must perform a rollforward operation to the end of logs.

If the damaged table space is *not* the system catalog table space, DB2 for Linux, UNIX, and Windows attempts to make as much of the database available as possible.

If the damaged table space is the only temporary table space, you should create a new temporary table space as soon as a connection to the database can be made. Once created, the new temporary table space can be used, and normal database operations requiring a temporary table space can resume. You can, if you want, drop the offline temporary table space. There are special considerations for table reorganization using a system temporary table space:

- If the database or the database manager configuration parameter **indexrec** is set to RESTART, all invalid indexes must be rebuilt during database activation; this includes indexes from a reorganization that crashed during the build phase.
- If there are incomplete reorganization requests in a damaged temporary table space, you might have to set the **indexrec** configuration parameter to ACCESS to avoid restart failures.

Recovering from transaction failures in a partitioned database environment

If a transaction failure occurs in a partitioned database environment, database recovery is usually necessary on both the failed database partition server and any other database partition server that was participating in the transaction.

There are two types of database recovery:

- Crash recovery occurs on the failed database partition server after the failure condition is corrected.
- *Database partition failure recovery* on the other (still active) database partition servers occurs immediately after the failure has been detected.

In a partitioned database environment, the database partition server on which a transaction is submitted is the coordinator partition, and the first agent that processes the transaction is the coordinator agent. The coordinator agent is responsible for distributing work to other database partition servers, and it keeps track of which ones are involved in the transaction. When the application issues a COMMIT statement for a transaction, the coordinator agent commits the transaction by using the two-phase commit protocol. During the first phase, the coordinator partition distributes a PREPARE request to all the other database partition servers that are participating in the transaction. These servers then respond with one of the following:

READ-ONLY
No data change occurred at this server

YES Data change occurred at this server

NO Because of an error, the server is not prepared to commit

If one of the servers responds with a `NO`, the transaction is rolled back. Otherwise, the coordinator partition begins the second phase.

During the second phase, the coordinator partition writes a COMMIT log record, then distributes a COMMIT request to all the servers that responded with a `YES`. After all the other database partition servers have committed, they send an acknowledgement of the COMMIT to the coordinator partition. The transaction is complete when the coordinator agent has received all COMMIT acknowledgments from all the participating servers. At this point, the coordinator agent writes a FORGET log record.

Transaction failure recovery on an active database partition server

If any database partition server detects that another server is down, all work that is associated with the failed database partition server is stopped:

- If the still active database partition server is the coordinator partition for an application, and the application was running on the failed database partition server (and not ready to COMMIT), the coordinator agent is interrupted to do failure recovery. If the coordinator agent is in the second phase of COMMIT processing, SQL0279N is returned to the application, which in turn loses its database connection. Otherwise, the coordinator agent distributes a ROLLBACK request to all other servers participating in the transaction, and SQL1229N is returned to the application.
- If the failed database partition server was the coordinator partition for the application, then agents that are still working for the application on the active servers are interrupted to do failure recovery. The transaction is rolled back locally on each database partition where the transaction is not in prepared state. On those database partitions where the transaction is in prepared state, the transaction becomes in doubt. The coordinator database partition is not aware that the transaction is in doubt on some database partitions because the coordinator database partition is not available.

- If the application connected to the failed database partition server (before it failed), but neither the local database partition server nor the failed database partition server is the coordinator partition, agents working for this application are interrupted. The coordinator partition will either send a ROLLBACK or a DISCONNECT message to the other database partition servers. The transaction will only be in doubt on database partition servers that are still active if the coordinator partition returns SQL0279.

Any process (such as an agent or deadlock detector) that attempts to send a request to the failed server is informed that it cannot send the request.

Transaction failure recovery on the failed database partition server

If the transaction failure causes the database manager to end abnormally, you can issue the **db2start** command with the RESTART option to restart the database manager once the database partition has been restarted. If you cannot restart the database partition, you can issue **db2start** to restart the database manager on a different database partition.

If the database manager ends abnormally, database partitions on the server can be left in an inconsistent state. To make them usable, crash recovery can be triggered on a database partition server:

- Explicitly, through the **RESTART DATABASE** command
- Implicitly, through a CONNECT request when the *autorestart* database configuration parameter has been set to `ON`

Crash recovery reapplies the log records in the active log files to ensure that the effects of all complete transactions are in the database. After the changes have been reapplied, all uncommitted transactions are rolled back locally, *except* for indoubt transactions. There are two types of indoubt transaction in a partitioned database environment:

- On a database partition server that is not the coordinator partition, a transaction is in doubt if it is prepared but not yet committed.
- On the coordinator partition, a transaction is in doubt if it is committed but not yet logged as complete (that is, the FORGET record is not yet written). This situation occurs when the coordinator agent has not received all the COMMIT acknowledgments from all the servers that worked for the application.

Crash recovery attempts to resolve all the indoubt transactions by doing one of the following. The action that is taken depends on whether the database partition server was the coordinator partition for an application:

- If the server that restarted is not the coordinator partition for the application, it sends a query message to the coordinator agent to discover the outcome of the transaction.
- If the server that restarted *is* the coordinator partition for the application, it sends a message to all the other agents (subordinate agents) that the coordinator agent is still waiting for COMMIT acknowledgments.

It is possible that crash recovery might not be able to resolve all the indoubt transactions. For example, some of the database partition servers might not be available. If the coordinator partition completes crash recovery before other database partitions involved in the transaction, crash recovery will not be able to resolve the indoubt transaction. This is expected because crash recovery is performed by each database partition independently. In this situation, the SQL warning message SQL1061W is returned. Because indoubt transactions hold resources, such as locks and active log space, it is possible to get to a point where no changes can be made to the database because the active log space is being held up by indoubt transactions. For this reason, you should determine whether indoubt transactions remain after crash recovery, and recover all database partition servers that are required to resolve the indoubt transactions as quickly as possible.

Note: In a partitioned database server environment, the RESTART database command is run on a per-node basis. In order to ensure that the database is restarted on all nodes, use the following recommended command:

```
db2_all "db2 restart database <database_name>"
```

If one or more servers that are required to resolve an indoubt transaction cannot be recovered in time, and access is required to database partitions on other servers, you can manually resolve the indoubt transaction by making an heuristic decision. You can use the **LIST INDOUBT TRANSACTIONS** command to query, commit, and roll back the indoubt transaction on the server.

Note: The **LIST INDOUBT TRANSACTIONS** command is also used in a distributed transaction environment. To distinguish between the two types of indoubt transactions, the *originator* field in the output that is returned by the **LIST INDOUBT TRANSACTIONS** command displays one of the following:

- DB2 Enterprise Server Edition, which indicates that the transaction originated in a partitioned database environment.
- XA, which indicates that the transaction originated in a distributed environment.

Identifying the failed database partition server

When a database partition server fails, the application will typically receive one of the following SQLCODEs. The method for detecting which database manager failed depends on the SQLCODE received:

SQL0279N
This SQLCODE is received when a database partition server involved in a transaction is terminated during COMMIT processing.

SQL1224N
This SQLCODE is received when the database partition server that failed is the coordinator partition for the transaction.

SQL1229N
This SQLCODE is received when the database partition server that failed is not the coordinator partition for the transaction.

Determining which database partition server failed is a two-step process.

1. Find the partition server that detected the failure by examining the SQLCA. The SQLCA associated with SQLCODE SQL1229N contains the node number of the server that detected the error in the sixth array position of the *sqlerrd* field. (The node number that is written for the server corresponds to the node number in the `db2nodes.cfg` file.)
2. Examine the administration notification log on the server found in step one for the node number of the failed server.

Note: If multiple logical nodes are being used on a processor, the failure of one logical node can cause other logical nodes on the same processor to fail.

Disaster recovery

The term *disaster recovery* is used to describe the activities that need to be done to restore the database in the event of a fire, earthquake, vandalism, or other catastrophic events.

A plan for disaster recovery can include one or more of the following:

- A site to be used in the event of an emergency
- A different machine on which to recover the database
- Offsite storage of either database backups, table space backups, or both, as well as archived logs.

If your plan for disaster recovery is to restore the entire database on another machine, it is recommended that you have at least one full database backup and all the archived logs for the database. Although it is possible to rebuild a database if you have a full table space backup of each table space in the database, this method might involve numerous backup images and be more time-consuming than recovery using a full database backup.

You can choose to keep a standby database up to date by applying the logs to it as they are archived. Or, you can choose to keep the database or table space backups and log archives in the standby site, and perform restore and rollforward operations only after a disaster has occurred. (In the latter case, recent backup images are preferable.) In a disaster situation, however, it is generally not possible to recover all of the transactions up to the time of the disaster.

The usefulness of a table space backup for disaster recovery depends on the scope of the failure. Typically, disaster recovery is less complicated and time-consuming if you restore the entire database; therefore, a full database backup should be kept at a standby site. If the disaster is a damaged disk, a table space backup of each table space on that disk can be used to recover. If you have lost access to a container because of a disk failure (or for any other reason), you can restore the container to a different location.

Another way you can protect your data from partial or complete site failures is to implement the DB2 high availability disaster recovery (HADR) feature. Once it is set up, HADR protects against data loss by replicating data changes from a source database, called the primary, to a target database, called the standby.

You can also protect your data from partial or complete site failures using replication. Replication allows you to copy data on a regular basis to multiple remote databases. DB2 database provides a number of replication tools that allow you to specify what data should be copied, which database tables the data should be copied to, and how often the updates should be copied.

Storage mirroring, such as Peer-to-Peer Remote Copy (PPRC), can also be used to protect your data. PPRC provides a synchronous copy of a volume or disk to protect against disasters.

DB2 database products provide you with several options when planning for disaster recovery. Based on your business needs, you might decide to use table space or full database backups as a safeguard against data loss, or you might decide that your environment is better suited to a solution like HADR. Whatever your choice, you should test your recovery procedures in a test environment before implementing them in your production environment.

Version recovery

Version recovery is the restoration of a previous version of the database, using an image that was created during a backup operation.

You use this recovery method with non-recoverable databases (that is, databases for which you do not have archived logs). You can also use this method with recoverable databases by using the **WITHOUT ROLLING FORWARD** option on the **RESTORE DATABASE** command.

A database restore operation will restore the entire database using a backup image created earlier. A database backup allows you to restore a database to a state identical to the one at the time that the backup was made. However, every unit of work from the time of the backup to the time of the failure is lost (see Figure 53).

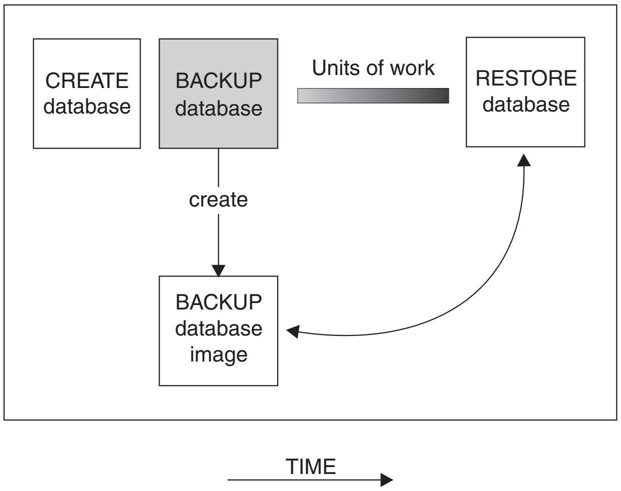


Figure 53. Version Recovery

Using the version recovery method, you must schedule and perform full backups of the database on a regular basis.

In a partitioned database environment, the database is located across many database partition servers (or nodes). You must restore all database partitions, and the backup images that you use for the restore database operation must all have been taken at the same time. (Each database partition is backed up and restored separately.) A backup of each database partition taken at the same time is known as a *version backup*.

Rollforward recovery

Use rollforward recover on a database or table space to recover to its state at a particular point in time, or to its state immediately before the failure.

To use the *rollforward recovery* method, you must have taken a backup of the database and archived the logs (by setting the **logarchmeth1** and **logarchmeth2** configuration parameters to a value other than OFF). Restoring the database and specifying the **WITHOUT ROLLING FORWARD** parameter is equivalent to using the version recovery method. The database is restored to a state identical to the one at the time that the offline backup image was made. If you restore the database and do *not* specify the **WITHOUT ROLLING FORWARD** parameter for the restore database operation, the database will be in rollforward pending state at the end of the restore operation. This allows rollforward recovery to take place.

Note: The **WITHOUT ROLLING FORWARD** parameter cannot be used if:

- You are restoring from an online backup image
- You are issuing a table space-level restore

During a recovery, archived log files are retrieved from the archive. If your archived log files are compressed, the files are automatically uncompressed and used. The archived log files are also automatically uncompressed when they are encountered in the active log path or overflow log path, if you manually copied the files there.

The two types of rollforward recovery to consider are:

- *Database rollforward recovery*. In this type of rollforward recovery, transactions recorded in database logs are applied following the database restore operation (see Figure 54). The database logs record all changes made to the database. This method completes the recovery of the database to its state at a particular point in time, or to its state immediately before the failure (that is, to the end of the active logs).

  In a partitioned database environment, the database is located across many database partitions, and the **ROLLFORWARD DATABASE** command must be issued on the database partition where the catalog tables for the database resides (catalog partition). If you are performing point-in-time rollforward recovery, all database partitions must be rolled forward to ensure that all database partitions are at the same level. If you need to restore a single database partition, you can perform rollforward recovery to the end of the logs to bring it up to the same level as the other database partitions in the database. Only recovery to the end of the logs can be used if one database partition is being rolled forward. Point-in-time recovery applies to *all* database partitions.

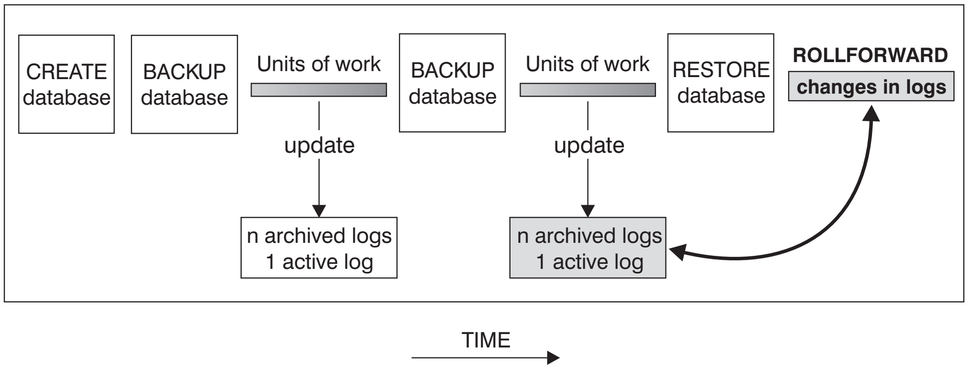


Figure 54. Database Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

- *Table space rollforward recovery*. If the database is enabled for forward recovery, it is also possible to back up, restore, and roll table spaces forward (see Figure 55 on page 732). To perform a table space restore and rollforward operation, you need a backup image of either the entire database (that is, all of the table spaces), or one or more individual table spaces. You also need the log records that affect the table spaces that are to be recovered. You can roll forward through the logs to one of two points:

- – The end of the logs; or,
- – A particular point in time (called *point-in-time recovery*).

Table space rollforward recovery can be used in the following two situations:

- After a table space restore operation, the table space is always in rollforward pending state, and it must be rolled forward. Invoke the **ROLLFORWARD DATABASE** command to apply the logs against the table spaces to either a point in time, or the end of the logs.
- If one or more table spaces are in *rollforward pending* state after crash recovery, first correct the table space problem. In some cases, correcting the table space problem does not involve a restore database operation. For example, a power loss could leave the table space in rollforward pending state. A restore database operation is not required in this case. Once the problem with the table space is corrected, you can use the **ROLLFORWARD DATABASE** command to apply the logs against the table spaces to the end of the logs. If the problem is corrected before crash recovery, crash recovery might be sufficient to take the database to a consistent, usable state.

  Note: If the table space in error contains the system catalog tables, you will not be able to start the database. You must restore the SYSCATSPACE table space, then perform rollforward recovery to the end of the logs.

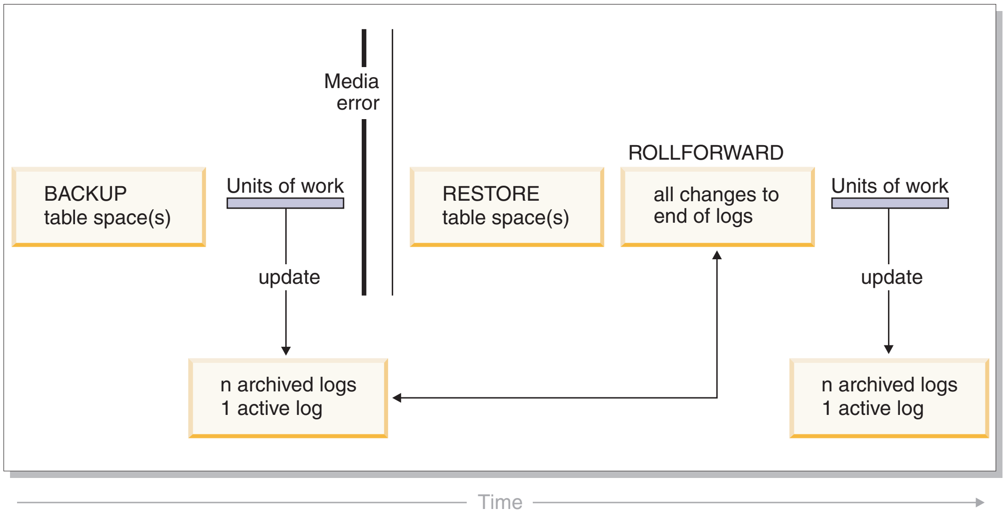


Figure 55. Table Space Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

In a partitioned database environment, if you are rolling a table space forward *to a point in time*, you do not have to supply the list of database partitions on which the table space resides. The DB2 database manager submits the rollforward request to all database partitions. This means the table space must be restored on all database partitions on which the table space resides.

In a partitioned database environment, if you are rolling a table space forward *to the end of the logs*, you must supply the list of database partitions if you do *not* want to roll the table space forward on all database partitions. If you want to roll all table spaces (on all database partitions) that are in rollforward pending state forward to the end of the logs, you do not have to supply the list of database partitions. By default, the database rollforward request is sent to all database partitions.

Table space rollforward operations behave differently in a DB2 pureScale environment. For more information, see “Log stream merging and log file management in a DB2 pureScale environment” on page 743 and “Log sequence numbers in DB2 pureScale environments” on page 747.

If you are rolling a table space forward that contains any piece of a partitioned table and you are rolling it forward to a point in time, you must also roll all of the other table spaces in which that table resides forward to the same point in time. However, you can roll a single table space containing a piece of a partitioned table forward to the end of logs.

If a partitioned table has any attached, detached, or dropped data partitions, then point-in-time rollforward must also include all table spaces for these data partitions. To determine if a partitioned table has any attached, detached, or dropped data partitions, query the SYSDATAPARTITIONS catalog table.

Storage group modifications during rollforward recovery

Whether storage group path modifications are redone during a rollforward operation depends on whether you redirected the storage group during the restore process. If you did not redefine a storage group during the database restore operation, log records affecting the storage group or its paths are replayed during rollforward recovery. Storage path updates, storage group rename operations, and table space storage group association updates that are described in the log records are applied during the rollforward operation. If a rollforward operation is attempting to replay a log record related to adding storage paths or creating a storage group and a storage path cannot be found, error SQL1051N is returned.

If you redefined storage paths during the restore operation, the rollforward operation does not redo any changes to storage paths or media attributes of storage groups whose paths you redirected. However, changes to the data tag or name of storage groups are redone. Also, log records for other operations, including DROP STOGROUP operations, are replayed. It is assumed that any explicitly specified storage group paths have been set to their desired final paths.

If a rebalance operation is encountered in the log, table space rebalance operations are initiated during rollforward recovery. The rebalance operations might not be completed while the rollforward operation is in progress. In that case, the rebalance processing is suspended at the completion of the rollforward operation and is restarted the next time that you activate the database.

During a rollforward operation, if a CREATE STOGROUP statement is encountered in the log, the storage group is created on the paths that you specified when you issued the CREATE STOGROUP statement.

Chapter 44. Developing a backup and recovery strategy

A database can become unusable because of hardware or software failure, or both. You might, at one time or another, encounter storage problems, power interruptions, or application failures, and each failure scenario requires a different recovery action.

Protect your data against the possibility of loss by having a well rehearsed recovery strategy in place.

Some of the questions that you should answer when developing your recovery strategy are:

- Will the database be recoverable?
- How much time can be spent recovering the database?
- How much time will pass between backup operations?
- How much storage space can be allocated for backup copies and archived logs?
- Will table space level backups be sufficient, or will full database backups be necessary?
- Should I configure a standby system, either manually or through high availability disaster recovery (HADR)?

A database recovery strategy should ensure that all information is available when it is required for database recovery. It should include a regular schedule for taking database backups and, in the case of partitioned database environments, include backups when the system is scaled (when database partition servers or nodes are added or dropped). Your overall strategy should also include procedures for recovering command scripts, applications, user-defined functions (UDFs), stored procedure code in operating system libraries, and load copies.

Different recovery methods are discussed in the sections that follow, and you will discover which recovery method is best suited to your business environment.

The concept of a database *backup* is the same as any other data backup: taking a copy of the data and then storing it on a different medium in case of failure or damage to the original. The simplest case of a backup involves shutting down the database to ensure that no further transactions occur, and then simply backing it up. You can then recreate the database if it becomes damaged or corrupted in some way.

The recreation of the database is called *recovery*. *Version recovery* is the restoration of a previous version of the database, using an image that was created during a backup operation. *Rollforward recovery* is the reapplication of transactions recorded in the database log files after a database or a table space backup image has been restored.

Crash recovery is the automatic recovery of the database if a failure occurs before all of the changes that are part of one or more units of work (transactions) are completed and committed. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.

Recovery log files and the recovery history file are created automatically when a database is created (Figure 56). These log files are important if you need to recover data that is lost or damaged.

Each database includes *recovery logs*, which are used to recover from application or system errors. In combination with the database backups, they are used to recover the consistency of the database right up to the point in time when the error occurred.

The *recovery history file* contains a summary of the backup information that can be used to determine recovery options, if all or part of the database must be recovered to a given point in time. It is used to track recovery-related events such as backup and restore operations, among others. This file is located in the database directory.

The *table space change history file*, which is also located in the database directory, contains information that can be used to determine which log files are required for the recovery of a particular table space.

You cannot directly modify the recovery history file or the table space change history file; however, you can delete entries from the files using the **PRUNE HISTORY** command. You can also use the **rec_his_retentn** database configuration parameter to specify the number of days that these history files will be retained.

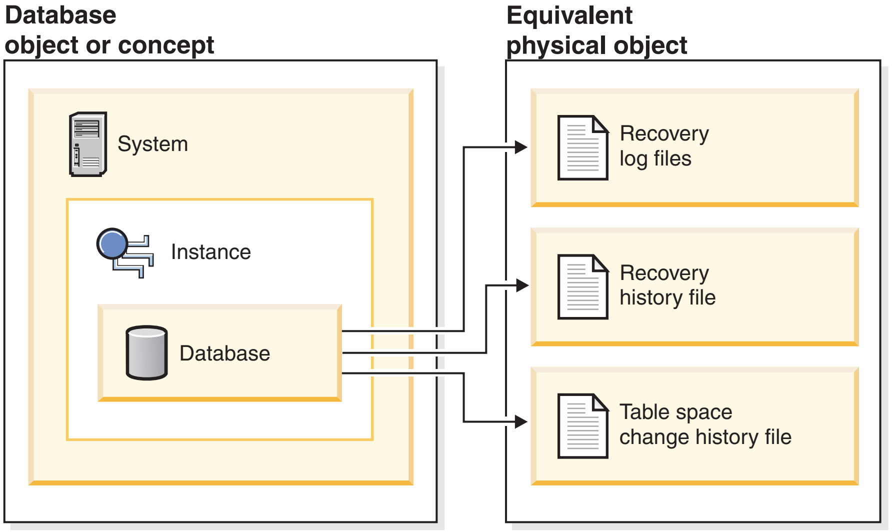


Figure 56. Database recovery files

Data that is easily re-created can be stored in a non-recoverable database. This includes data from an outside source that is used for read-only applications, and tables that are not often updated, for which the small amount of logging does not justify the added complexity of managing log files and rolling forward after a restore operation. If both the **logarchmeth1** and **logarchmeth2** database configuration parameters are set toOFF then the database is *Non-recoverable*. This means that the only logs that are kept are those required for crash recovery. These logs are known as *active logs*, and they contain current transaction data. Version recovery using *offline* backups is the primary means of recovery for a non-recoverable database. (An offline backup means that no other application can use the database when the backup operation is in progress.) Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken and rollforward recovery is not supported.

Data that *cannot* be easily recreated should be stored in a recoverable database. This includes data whose source is destroyed after the data is loaded, data that is manually entered into tables, and data that is modified by application programs or users after it is loaded into the database. *Recoverable databases* have the **logarchmeth1** or **logarchmeth2** database configuration parameters set to a value other than OFF. Active logs are still available for crash recovery, but you also have the *archived logs*, which contain committed transaction data. Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken. However, with rollforward recovery, you can roll the database forward (that is, past the time when the backup image was taken) by using the active and archived logs to either a specific point in time, or to the end of the active logs.

Recoverable database backup operations can be performed either offline or *online* (online meaning that other applications can connect to the database during the backup operation). Online table space restore and rollforward operations are supported only if the database is recoverable. If the database is non-recoverable, database restore and rollforward operations must be performed offline. During an online backup operation, rollforward recovery ensures that *all* table changes are captured and reapplied if that backup is restored.

If you have a recoverable database, you can back up, restore, and roll individual table spaces forward, rather than the entire database. When you back up a table space online, it is still available for use, and simultaneous updates are recorded in the logs. When you perform an online restore or rollforward operation on a table space, the table space itself is not available for use until the operation completes, but users are not prevented from accessing tables in other table spaces.

Automated backup operations

Since it can be time-consuming to determine whether and when to run maintenance activities such as backup operations, you can use automatic maintenance. With automatic maintenance, you specify your maintenance objectives, including when automatic maintenance can run. DB2 then uses these objectives to determine if the maintenance activities need to be done and then runs only the required maintenance activities during the next available maintenance window (a user-defined time period for the running of automatic maintenance activities).

Note: You can still perform manual backup operations when automatic maintenance is configured. DB2 will only perform automatic backup operations if they are required.

Deciding how often to back up

Your recovery plan should allow for regularly scheduled backup operations, because backing up a database requires time and system resources. Your plan might include a combination of full database backups and incremental backup operations. Also, the frequency and types of backups you make affect your database recovery time.

Take full database backups regularly, even if you archive the logs to allow for rollforward recovery. To recover a database, you can use either a full database backup image that contains all of the table space backup images, or you can rebuild the database by using selected table space images. Table space backup images are also useful for recovering from an isolated disk failure or an application error. In partitioned database environments, you need to restore only the table spaces that reside on database partitions that failed. You do not need to restore all of the table spaces or all of the database partitions.

Although full database backups are no longer required for database recovery because you can rebuild a database from table space images, it is still good practice to occasionally take a full backup of your database.

You should also consider not overwriting backup images and logs, saving at least two full database backup images and their associated logs as an extra precaution.

If the amount of time needed to apply archived logs when recovering and rolling an active database forward is a major concern, consider the cost of backing up the database more frequently. More frequent backups reduce the number of archived logs you need to apply when rolling forward.

Online and offline backup considerations

You can initiate a backup operation while the database is either online or offline. If it is online, other applications or processes can connect to the database, as well as read and modify data while the backup operation is running. If the backup operation is running offline, other applications cannot connect to the database.

To reduce the amount of time that the database is not available, consider using online backup operations. Online backup operations are supported only if rollforward recovery is enabled. If rollforward recovery is enabled and you have a complete set of recovery logs, you can restore the database, should the need arise. You can use an online backup image for recovery only if you have the logs that span the time during which the backup operation was running.

Offline backup operations are faster than online backup operations, since there is no contention for the data files.

Selective table space backup considerations

You can use the backup utility to back up only selected table spaces. If you use DMS table spaces, you can store different types of data in their own table spaces to reduce the time required for backup operations. You can keep table data in one table space, long field and LOB data in another table space, and indexes in yet another table space. If you separate your data into different table spaces and a disk failure occurs, the disk failure is likely to affect only one of the table spaces. Restoring or rolling forward one of these table spaces takes less time than it would take to restore a single table space that contains all of the data.

You can also save time by taking backups of different table spaces at different times, as long as the changes to them are not the same. So, if long field or LOB data is not changed as frequently as the other data, you can back up these table spaces less frequently. If long field and LOB data are not required for recovery, you can also consider not backing up the table space that contains that data. If the LOB data can be reproduced from a separate source, choose the NOT LOGGED option when creating or altering a table to include LOB columns.

If you keep your long field data, LOB data, and indexes in separate table spaces, but do not back them up together, consider the following point: If you back up a table space that does not contain all of the table data, you cannot perform point-in-time rollforward recovery on that table space. All the table spaces that contain any type of data for a table must be rolled forward simultaneously to the same point in time.

Table reorganization considerations

If you reorganize a table, you should back up the affected table spaces after the operation completes. If you have to restore the table spaces, you will not have to roll forward through the data reorganization.

Table space modification status considerations

You can also make more informed decisions about whether to back up a table space by checking its modification status. The **db2pd -tablespaces trackmodstate** command and the **tbsp_trackmode_state** monitor element displays the status of the table space with respect to the last or next backup. You can use this information to determine whether the table space was modified or if the table space needs to be backed up.

Database recovery time considerations

The time required to recover a database is made up of two parts:

- The time required to complete the restoration of the backup.
- If the database is enabled for forward recovery, the time required to apply the logs during the rollforward operation

When formulating a recovery plan, take these recovery costs and their impact on your business operations into account. Testing your overall recovery plan assists you in determining whether the time required to recover the database is reasonable, given your business requirements. Following each test, you might want to increase the frequency with which you take a backup. If rollforward recovery is part of your strategy, this increased backup frequency reduces the number of logs that are archived between backups and, as a result, reduces the time required to roll the database forward after a restore operation.

Storage considerations for recovery

When deciding which recovery method to use, consider the storage space required. Backup and archived log file compression can help reduce the storage cost in your database environment.

The version recovery method requires space to hold the backup copy of the database and the restored database. The roll-forward recovery method requires space to hold the backup copy of the database or table spaces, the restored database, and the archived database logs.

If a table contains long field or large object (LOB) columns, you might consider placing this data into a separate table space. This action affects your storage space considerations, as well as affect your plan for recovery. With a separate table space for long field and LOB data, and knowing the time required to back up long field and LOB data, you might decide to use a recovery plan that only occasionally saves a backup of this table space. You can also choose, when creating or altering a table to include LOB columns, not to log changes to those columns. This action reduces the size of the required log space and the corresponding archived log file space.

To prevent media failure from destroying a database and your ability to restore it, keep the database backup, the database logs, and the database itself on different devices. For this reason, it is highly recommended that you use the *newlogpath* configuration parameter to put database logs on a separate device once the database is created.

The database logs can use up a large amount of storage. If you plan to use the roll-forward recovery method, you must decide how to manage and compress the archived logs. Your choices are:

- Specify an archived log file method using the LOGARCHMETH1 or LOGARCHMETH2 configuration parameters.
- Enable archived log file compression with the LOGARCHCOMPR1 and LOGARCHCOMPR2 configuration parameters.
- Manually copy the logs to a storage device or directory other than the database log path directory after they are no longer in the active set of logs.
- Use a user exit program to copy these logs to another storage device in your environment.

Backup compression

In addition to the storage savings you can achieve through row compression in your active database, you can also use backup compression to reduce the size of your database backups.

Whereas row compression works on a table-by-table basis, when you use compression for your backups, *all* of the data in the backup image is compressed, including catalog tables, index objects, LOB objects, auxiliary database files and database meta-data.

You can use backup compression with tables that use row compression. Keep in mind, however, that backup compression requires additional CPU resources and extra time. It may be sufficient to use table compression alone to achieve a reduction in your backup storage requirements. If you are using row compression, consider using backup compression only if storage optimization is of higher priority than the extra time it takes to perform the backup.

Tip: Consider using backup compression only on table spaces that do not contain compressed data if the following conditions apply:

- Data and index objects are separate from LOB and long field data, and
- You use row and index compression on the majority of your data tables and indexes, respectively

To use compression for your backups, use the `COMPRESS` option on the **`BACKUP DATABASE`** command.

Archived log file compression

As of DB2 V10.1, you can compress archived log files. This capability, in addition to data and index compression, along with backup compression, reduces the amount of disk space required for your database environment.

Archived log files are the third major space consumer for roll-forward recoverable databases. Archived log files contain a significant amount of data and these archives can grow quickly. If modified data is already in compressed tables, logging is reduced by virtue of including compressed record images in log records. Compression of archived log files further increases storage savings, even in these environments.

To use compression for your archived log files, you can use the **UPDATE DB CFG** command to set the **logarchcompr1** and **logarchcompr2** configuration parameters to ON.

Restrictions

- Archived log file compression does not take effect under the following conditions.
  - The corresponding archived log file method is not set to DISK, TSM, or VENDOR. When the corresponding archived log file method is set as described, the log files are physically moved out of the active log path, or the mirror log path.
  - Whenever archived log file compression is enabled, but the corresponding log archiving method is set to OFF, LOGRETAIN or USEREXIT, archived log file compression has no effect. Any update to the **logarchmeth1** and **logarchmeth2** or the **logarchcompr1** and **logarchcompr2** database configuration parameters which results in such a scenario returns a warning, SQL1663W.

    Note: When the database is activated, SQL1663W is not returned when setting or changing archived log file compression database configuration parameters. Instead, SQL1363W is returned, which is a higher priority message. If the database is not activated, the SQL1663W warning message is returned.
- Manual archiving and retrieval with **db2adutl**.
  - The **db2adutl** utility does not perform compression or decompression during UPLOAD or EXTRACT operations. Movement of compressed log files to and from the archive location is fully supported by **db2adutl**.
  - If logs are uploaded to Tivoli Storage Manager with **db2adutl**, and you want to compress archived log files, archived log file compression must be enabled when the logs are archived to the disk location, before **db2adutl** picks them up. If compressed logs are retrieved manually with **db2adutl**, they are extracted on first access.
- Archived log file compression is not supported when raw devices are used for database logging.
  - Archived log file compression is not supported when either the **logpath** or the **newlogpath** database configuration parameters point to a raw device. Any database configuration update that results in archived log file compression being enabled while **logpath** or **newlogpath** database configuration parameters point to raw devices fails, SQL1665N.
- When enabling archived log file compression using the **logarchcompr1** and **logarchcompr2** database configuration parameters, logs already stored in a backup image are not affected.

Backup and restore operations between different operating systems and hardware platforms

DB2 database systems support some backup and restore operations between different operating systems and hardware platforms.

The supported platforms for DB2 backup and restore operations can be grouped into one of three families:

- Big-endian Linux and UNIX
- Little-endian Linux and UNIX
- Windows

A database backup from one platform family can only be restored on any system within the same platform family. For Windows operating systems, you can restore a database that was created on DB2 Version 9.7 on a DB2 Version 10.1 database system. For Linux and UNIX operating systems, as long as the endianness (big endian or little endian) of the backup and restore platforms is the same, you can restore backups that were produced on down level versions.

The following table shows each of the Linux and UNIX platforms DB2 supports and indicates whether the platforms are big endian or little endian:

Table 122. Endianness of supported Linux and UNIX operating systems DB2 supports

Platform	**Endianness**
AIX	big endian
HP on IA64	big endian
Solaris x64	little endian
Solaris SPARC	big endian
Linux on zSeries	big endian
Linux on pSeries	big endian
Linux on IA-64	little endian
Linux on AMD64 and Intel EM64T	little endian
32-bit Linux on x86	little endian

The target system must have the same (or later) version of the DB2 database product as the source system. You cannot restore a backup that was created on one version of the database product to a system that is running an earlier version of the database product. For example, you can restore a DB2 Version 9.7 on a DB2 Version 10.1 database system, but you cannot restore a DB2 Version 10.1 backup on a DB2 Version 9.7 database system.

Note: You can restore a database from a backup image that was taken on a 32-bit level into a 64-bit level, but not vice versa. The DB2 backup and restore utilities should be used to back up and restore your databases. Moving a file set from one machine to another is not recommended as this can compromise the integrity of the database.

In situations where certain backup and restore combinations are not allowed, you can move tables between DB2 databases using other methods:

- The **db2move** command
- The **export** command followed by the **import** or the **load** command

Note: Database configuration parameters are set to their defaults if the values in the backup are outside of the allowable range for the environment in which the database is being restored.

Log stream merging and log file management in a DB2 pureScale environment

In a DB2 pureScale environment, each member maintains its own set of transaction log files (that is, a *log stream*) on the shared disk, each set in a separate log path. The log files for a member contain a history of all data changes that occurred on that member.

Multiple applications, each accessing a different member simultaneously, might generate dependent transactions during run time. A dependency between two transactions can occur if, for example, both transactions change the same row. To effectively interpret the log records, the DB2 data server must examine the records from all log streams and order the records so that they reflect the order of the updates that occurred at run time. This ordering is known as a *log stream merge* operation. Several operation types in a DB2 pureScale environment require log stream merges; these include (among others) group crash recovery, database roll-forward operations, and table space roll-forward operations.

Logging configuration parameters in a DB2 pureScale environment

Table 123 shows which logging-related database configuration parameters are global in scope and which parameters are dynamically updatable.

Table 123. Logging-related database configuration parameters

Parameter	Global?	Dynamically updatable?
archretrydelay	Yes	Yes
blk_log_dsk_ful	No	Yes
failarchpath	Yes	Yes
logarchcompr1	Yes	Yes
logarchcompr2	Yes	Yes
logarchmeth1	Yes	Yes
logarchmeth2	Yes	Yes
logarchopt1	Yes	Yes
logarchopt2	Yes	Yes
logbufsz	No	Yes
logfilsiz	Yes	No
logprimary	Yes	No
logsecond	Yes	Yes
max_log	No	Yes
mirrorlogpath [1]	Yes	No
newlogpath [1]	Yes	No
num_log_span	No	Yes
numarchretry	Yes	Yes
overflowlogpath	Yes	Yes
softmax	Yes	No
vendoropt	Yes	Yes

Table 123. Logging-related database configuration parameters (continued)

Parameter	Global?	Dynamically updatable?
[1] The first member that connects to or activates the database processes the changes to this log path parameter. The DB2 database manager verifies that the path exists and that it has both read and write access to that path. It also creates member-specific subdirectories for the log files. If any one of these operations fails, the DB2 database manager rejects the specified path and brings the database online using the old path. If the database manager accepts the specified path, the new value is propagated to each member. If a member fails while trying to switch to the new path, subsequent attempts to activate the database or to connect to it fails, and SQL5099N is returned. All members must use the same log path.		

Retrieving logs for a log stream merge operation in a DB2 pureScale environment

A subdirectory is created in the path for retrieved log files. The subdirectory has the following format: *log_path*/LOGSTREAM*xxxx*, where *log_path* represents the log path, overflow log path, or mirror log path, and *xxxx* is a 4-digit log stream identifier. (The log stream identifier is not necessarily equivalent to the associated member ID.) Within this subdirectory, if a member requires log retrieval, the DB2 database manager creates another level of subdirectories for retrieved logs from each member. For example, if you specify an overflow log path of `/home/dbuser/overflow/` on a 3-member system, and an application on member 0 must retrieve logs that are owned by other members, the path for member 0 is `/home/dbuser/overflow/NODE0000/LOGSTREAM0000`, and subdirectories under this path contain retrieved logs that are owned by other members, as shown in the following example:

```
Member 0 retrieves its own logs here:
   /home/dbuser/overflow/NODE0000/LOGSTREAM0000/LOGSTREAM0000
Member 0 retrieves logs that belong to member 1 here:
   /home/dbuser/overflow/NODE0000/LOGSTREAM0000/LOGSTREAM0001
Member 0 retrieves logs that belong to member 2 here:
   /home/dbuser/overflow/NODE0000/LOGSTREAM0000/LOGSTREAM0002
```

Note: Do not manually insert log files in to these retrieve subdirectories. If you want to manually retrieve log files, use the overflow log path instead.

When reading archived log files that are owned by other members, a member might need to retrieve log files in to its own log path or overflow log path. In this case, the log stream merge operation creates a **db2logmgr** engine dispatchable unit (EDU) for each log stream, as needed.

As mentioned earlier, there are three paths that can be used to store log files that are owned by other members, as shown in the following list:

1. If you set the **overflowlogpath** database configuration parameter, the overflow log path is used.

   Tip: You can use **ROLLFORWARD DATABASE** and **RECOVER DATABASE** command options to specify an alternative overflow log path; the values of these options override the database configuration for purposes of the single recovery operation.
2. The primary log path
3. If you set the **mirrorlogpath** database configuration parameter, the mirror log path is used.

If the DB2 database manager is unable to store a log file in the first path, it attempts to use the next path in the list. If none of these paths is available, the utility that invoked the log stream merge operation returns an error that is specific to that utility.

Output from the **GET DATABASE CONFIGURATION** command in a DB2 pureScale environment identifies each log path followed by the name of the member. For example, if the mirror log path was set to /home/dbuser/mirrorpath/, for member 2, the output displays /home/dbuser/mirrorpath/NODE0000/LOGSTREAM0002.

If you must manually retrieve log files that are owned by other members, ensure that the database manager can access the log files by using the same directory structure that is automatically created. For example, to make logs from member 2 available in the overflow log path of member 1, place the logs in the /home/dbuser/overflow/NODE0000/LOGSTREAM0001/LOGSTREAM0002 directory.

Retrieved log files are automatically deleted when they are no longer needed. Subdirectories that were created during a log stream merge operation are retained for future use.

Detection of missing logs during a log stream merge operation

If you accidentally deleted, moved, or archived and lost a log file that is required for a recovery operation, you can roll-forward recover the database to the last consistent point before the missing log file.

If, during a log stream merge operation, the DB2 database manager determines that there is a missing log file in one of the log streams, an error is returned. The roll-forward utility returns SQL1273N; the db2ReadLog API returns SQL2657N.

Figure 57 shows an example of how two members could write log records to the log files in their active log stream. Each log file is represented by a box.

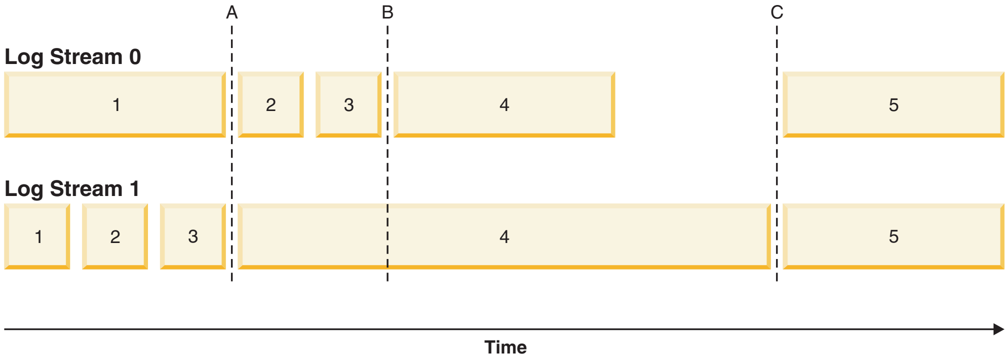


Figure 57. Log files in a DB2 pureScale environment

Consider a scenario where only log file 4 from log stream 1 is missing, a roll-forward operation to time A succeeds while roll-forward operations to time B, time C, or to the END OF LOGS fail. The ROLLFORWARD command returns SQL1273N because log file 4 is not available. Furthermore, since the log records in files 2 and 3 on log stream 0 were written during the same time period as the beginning of log file 4 on log stream 1, the roll-forward operation cannot process log files 2 and 3 until log file 4 from log stream 1 is available. The result is that the roll-forward operation stops at time A, and any subsequent roll-forward operations cannot proceed beyond time A until log 4 from stream 1 becomes available.

Consider another scenario where only log file 4 from log stream 0 is missing during a roll-forward operation. If you issue a **ROLLFORWARD** command with the **END OF LOGS** option (or anytime after time B), the operation will stop at time B and will return SQL1273N because log file 4 on stream 0 is missing. A roll-forward operation can replay log records from files 2 and 3 on log stream 0 and some logs from file 4 on stream 1 up to time B. The roll-forward operation must stop at time B even though additional logs from stream 1 are available because the log merge process requires that all the logs from all the streams be available.

If you can find the missing log file, make it available and reissue the **ROLLFORWARD DATABASE** command. If you cannot find the missing log file, issue the **ROLLFORWARD DATABASE...STOP** command to complete the roll-forward operation at the last consistent point just before the missing log file.

Although missing log detection ensures that database corruption does not occur as a result of missing log files, the presence of missing log files prevents some transactions from being replayed and, as a result, data loss could occur if the missing log files are not located.

Required resources

Log stream merge operations require additional EDUs. During database activation, one **db2lfr** EDU is created on each member. When a log read operation that requires a log stream merge is initiated, one **db2shred** EDU and one **db2lfr** EDU is created for each log stream. Although each **db2lfr**-**db2shred** group allocates its own set of log page and log record buffers, this is not a significant amount of additional memory or system resources; approximately 400 KB is allocated for each member that is involved in the log stream merge.

During a log stream merge operation, a member retrieves log files that are owned by other members into its overflow log path, primary log path, or mirror log path. In a DB2 pureScale environment, ensure that there is adequate free disk space in the retrieval path before starting a roll-forward operation. This allows the operation to retrieve the larger number of files from the archive, as required in a DB2 pureScale environment, without affecting performance. Use the following rule-of-thumb to calculate how much space you need to retrieve the active log files for all members: (**logprimary** + **logsecond**) * number of members.

Examples

- Update the **newlogpath** global database configuration parameter:

  ```
  db2 update db cfg for db mydb using newlogpath /home/dbuser/logdir
  ```

- Update the **max_log** per-member database configuration parameter on a single member:

  ```
  db2 update db cfg for db mydb member 1 using max_log 5
  ```

- Update the primary log path:

  ```
  db2 connect to mydb
  db2 update db cfg for mydb using newlogpath /home/dbuser/newlogpath
  db2 get db cfg for mydb
  ...
  Changed path to log files (NEWLOGPATH) = /home/dbuser/newlogpath/NODE0000/LOGSTREAM0000/
  Path to log files                      = /home/dbuser/dbuser/NODE0000/LOGSTREAM0000/
  ...
  ```

  The change does not take effect because the member is still active.

```
db2 terminate
db2 deactivate db mydb
db2 connect to mydb
db2 get db cfg for mydb
...
Changed path to log files (NEWLOGPATH) =
Path to log files                      = /home/dbuser/newlogpath/NODE0000/LOGSTREAM0000/
...
```

Each member uses the /home/dbuser/newlogpath/NODE0000/LOGSTREAM*xxxx* log path, where *xxxx* is the log stream ID of the log stream that uses the path.

- Set a new primary log path while restoring a backup image:

```
db2 restore db mydb newlogpath '/home/dbuser/newlogpath' without prompting
```

Log sequence numbers in DB2 pureScale environments

DB2 databases use the log sequence number (LSN), a 64-bit identifier, to determine the order of the operations that generated the log records.

The LSN is an ever-increasing value. Each member writes to its own set of log files (a *log stream*), and the LSN within a single log stream is a unique number.

Because LSNs are generated independently on each member and there are multiple log streams, it is possible to have duplicate LSN values across different log streams. A log record identifier (LRI) is used to identify log records across log streams; each log record in any log stream in the database is assigned a unique LRI. Use the **db2pd** command to determine which LRI is being processed by a recovery operation.

Including log files with a backup image

When performing an online backup operation, you can specify that the log files required to restore and recover a database are included in the backup image.

This means that if you need to ship backup images to a disaster recovery site, you do not have to send the log files separately or package them together yourself. Further, you do not have to decide which log files are required to guarantee the consistency of an online backup. This provides some protection against the deletion of log files required for successful recovery.

To use this feature, specify the **INCLUDE LOGS** option of the **BACKUP DATABASE** command. When you specify this option, the backup utility truncates the currently active log file and copies the necessary set of log extents into the backup image.

To restore the log files from a backup image, use the **LOGTARGET** option of the **RESTORE DATABASE** command and specify a fully qualified path that exists on the DB2 server. The restore database utility then writes the log files from the image to the target path. If a log file with the same name exists in the target path, the restore operation fails and an error is returned. If the **LOGTARGET** option is not specified, no log files are restored from the backup image.

If the **LOGTARGET** option is specified and the backup image does not include any log files, an error is returned before an attempt is made to restore any table space data. The restore operation also fails if an invalid or read-only path is specified. During a database or table space restore where the **LOGTARGET** option is specified, if one or more log files cannot be extracted, the restore operation fails and an error is returned.

You can also choose to restore only the log files saved in the backup image. To do this, specify the **LOGS** option with the **LOGTARGET** option of the **RESTORE DATABASE** command. If the restore operation encounters any problems when restoring log files in this mode, the restore operation fails and an error is returned.

During an automatic incremental restore operation, only the logs included in the target image of the restore operation are retrieved from the backup image. Any logs that are included in intermediate images referenced during the incremental restore process are not extracted from those backup images. During a manual incremental restore, if you specify a log target directory when restoring a backup image that includes log files, the log files in that backup image are restored.

If you roll a database forward that was restored from an online backup image that includes log files, you might encounter error SQL1268N, which indicates roll-forward recovery stopped due to an error received when retrieving a log. This error is generated when the target system to which you are attempting to restore the backup image does not have access to the facility used by the source system to archive its transaction logs.

If you specify the **INCLUDE LOGS** option of the **BACKUP DATABASE** command when you back up a database, then perform a restore operation and a roll-forward operation that use that back up image, DB2 still searches for additional transaction logs when rolling the database forward, even though the backup image includes logs. It is standard rollforward behavior to continue to search for additional transaction logs until no more logs are found. It is possible to have more than 1 log file with the same timestamp. Consequently, DB2 does not stop as soon as it finds the first timestamp that matches the point-in-time to which you are rolling forward the database as there might be other log files that also have that timestamp. Instead, DB2 continues to look at the transaction log until it finds a timestamp greater than the point-in-time specified.

When no additional logs can be found, the rollforward operation ends successfully. However, if there is an error while searching for additional transaction log files, error SQL1268N is returned. Error SQL1268N can occur because during the initial restore, certain database configuration parameters were reset or overwritten. Three of these database configuration parameters are the TSM parameters, **tsm_nodename**, **tsm_owner**, and **tsm_password**. They are all reset to NULL. To rollforward to the end of logs, you need to reset these database configuration parameters to correspond to the source system before the rollforward operation. Alternatively, you can specify the **NORETRIEVE** option when you issue the **ROLLFORWARD DATABASE** command. This prevents the DB2 database system from trying to obtain potentially missing transaction logs elsewhere.

Note:

1. This feature is not supported for offline backups.
2. When logs are included in an online backup image, the resulting image cannot be restored on releases of DB2 database before Version 8.2.

Incremental backup and recovery

As the size of databases, and particularly warehouses, continues to expand into the terabyte and petabyte range, the time and hardware resources required to back up and recover these databases is also growing substantially.

Full database and table space backups are not always the best approach when dealing with large databases, because the storage requirements for multiple copies of such databases are enormous.

Consider the following issues:

- When a small percentage of the data in a warehouse changes, it should not be necessary to back up the entire database.
- Appending table spaces to existing databases and then taking only table space backups is risky, because there is no guarantee that nothing outside of the backed up table spaces has changed between table space backups.

To address these issues, DB2 provides incremental backup and recovery.

An *incremental backup* is a backup image that contains only pages that have been updated since the previous backup was taken. In addition to updated data and index pages, each incremental backup image also contains all of the initial database metadata (such as database configuration, table space definitions, database history, and so on) that is normally stored in full backup images.

Note:

1. If a table space contains long field or large object data and an incremental backup is taken, all of the long field or large object data will be copied into the backup image if any of the pages in that table space have been modified since the previous backup.
2. If you take an incremental backup of a table space that contains a dirty page (that is, a page that contains data that has been changed but has not yet been written to disk) then all large object data is backed up. Normal data is backed up only if it has changed.
3. Data redistribution might create table spaces for all new database partitions if the **ADD DBPARTITIONNUMS** parameter on the **REDISTRIBUTE DATABASE PARTITION GROUP** command is specified; this can affect incremental backup operations.

Two types of incremental backup are supported:

- *Incremental*. An incremental backup image is a copy of all database data that has changed since the most recent, successful, full backup operation. This is also known as a cumulative backup image, because a series of incremental backups taken over time will each have the contents of the previous incremental backup image. The predecessor of an incremental backup image is always the most recent successful full backup of the same object.
- *Delta*. A delta, or incremental delta, backup image is a copy of all database data that has changed since the last successful backup (full, incremental, or delta) of the table space in question. This is also known as a differential, or noncumulative, backup image. The predecessor of a delta backup image is the most recent successful backup containing a copy of each of the table spaces in the delta backup image.

The key difference between incremental and delta backup images is their behavior when successive backups are taken of an object that is continually changing over time. Each successive incremental image contains the entire contents of the previous incremental image, plus any data that has changed, or is new, since the previous full backup was produced. Delta backup images contain only the pages that have changed since the previous image of any type was produced.

Combinations of database and table space incremental backups are permitted, in both online and offline modes of operation. Be careful when planning your backup strategy, because combining database and table space incremental backups implies that the predecessor of a database backup (or a table space backup of multiple table spaces) is not necessarily a single image, but could be a unique set of previous database and table space backups taken at different times.

To restore the database or the table space to a consistent state, the recovery process must begin with a consistent image of the entire object (database or table space) to be restored, and must then apply each of the appropriate incremental backup images in the order described in the following list.

To enable the tracking of database updates, DB2 supports a new database configuration parameter, **trackmod**, which can have one of two accepted values:

- NO. Incremental backup is not permitted with this configuration. Database page updates are not tracked or recorded in any way. This is the default value.
- YES. Incremental backup is permitted with this configuration. When update tracking is enabled, the change becomes effective at the first successful connection to the database. Before an incremental backup can be taken on a particular table space, a full backup of that table space is necessary.

For SMS and DMS table spaces, the granularity of this tracking is at the table space level. In table space level tracking, a flag for each table space indicates whether or not there are pages in that table space that need to be backed up. If no pages in a table space need to be backed up, the backup operation can skip that table space altogether.

Although minimal, the tracking of updates to the database can have an impact on the runtime performance of transactions that update or insert data.

Restoring from incremental backup images

A restore operation from incremental backup images consists of four steps.

About this task

1. Identifying the incremental target image.

   Determine the final image to be restored, and request an incremental restore operation from the DB2 restore utility. This image is known as the target image of the incremental restore, because it is the last image to be restored. The incremental target image is specified using the **TAKEN AT** parameter in the **RESTORE DATABASE** command.
2. Restoring the most recent full database or table space image to establish a baseline against which each of the subsequent incremental backup images can be applied.
3. Restoring each of the required full or table space incremental backup images, in the order in which they were produced, on top of the baseline image restored in Step 2.
4. Repeating Step 3 until the target image from Step 1 is read a second time. The target image is accessed twice during a complete incremental restore operation. During the first access, only initial data is read from the image; none of the user data is read. The complete image is read and processed only during the second access.

   The target image of the incremental restore operation must be accessed twice to ensure that the database is initially configured with the correct history, database configuration, and table space definitions for the database that is created during the restore operation. In cases where a table space was dropped since the initial full database backup image was taken, the table space data for that image is read from the backup images but ignored during incremental restore processing.

There are two ways to restore incremental backup images: automatic and manual:

- For an automatic incremental restore, the **RESTORE DATABASE** command is issued only once specifying the target image to be used. DB2 for Linux, UNIX, and Windows then uses the database history to determine the remaining required backup images and restores them.
- For a manual incremental restore, the **RESTORE DATABASE** command must be issued once for each backup image that needs to be restored (as outlined in the steps listed previously).

Procedure

- To restore a set of incremental backup images using automatic incremental restore, issue the **RESTORE DATABASE** command specifying time stamp of the last image you want to restore with the **TAKEN AT** parameter, as follows:

  ```
  db2 restore db sample incremental automatic taken at timestamp
  ```

  This results in the restore utility performing each of the steps described at the beginning of this section automatically. During the initial phase of processing, the backup image with the specified time stamp (specified in the form *yyyymmddhhmmss*) is read, and the restore utility verifies that the database, its history, and the table space definitions exist and are valid.

  During the second phase of processing, the database history is queried to build a chain of backup images required to perform the requested restore operation. If, for some reason this is not possible, and DB2 for Linux, UNIX, and Windows is unable to build a complete chain of required images, the restore operation terminates, and an error message is returned. In this case, an automatic incremental restore is not possible, and you must issue the **RESTORE DATABASE** command with the **INCREMENTAL ABORT** parameter. This will clean up any remaining resources so that you can proceed with a manual incremental restore.

  Note: It is highly recommended that you not use the **WITH FORCE OPTION** of the **PRUNE HISTORY** command. The default operation of this command prevents you from deleting history entries that might be required for recovery from the most recent, full database backup image, but with the **WITH FORCE OPTION**, it is possible to delete entries that are required for an automatic restore operation.

  During the third phase of processing, DB2 for Linux, UNIX, and Windows restores each of the remaining backup images in the generated chain. If an error occurs during this phase, you must issue the **RESTORE DATABASE** command with the **INCREMENTAL ABORT** option to clean up any remaining resources. You must then determine whether the error can be resolved before you reissue the **RESTORE DATABASE** command or attempt the manual incremental restore again.
- To restore a set of incremental backup images, using manual incremental restore, issue **RESTORE DATABASE** commands specifying time stamp of each image you want to restore with the **TAKEN AT** parameter, as follows:

  1.

     ```
     db2 restore database dbname incremental taken at timestamp
     ```

     where *timestamp* points to the last incremental backup image (*the target image*) to be restored.

2.

   ```
   db2 restore database dbname incremental taken at timestamp1
   ```

   where *timestamp1* points to the initial full database (or table space) image.
3.

   ```
   db2 restore database dbname incremental taken at timestampX
   ```

   where *timestampX* points to each incremental backup image in creation sequence.
4.

   Repeat Step 3, restoring each incremental backup image up to and including image *timestamp*.

If you are performing a database restore operation, and table space backup images have been produced, the table space images must be restored in the chronological order of their backup time stamps.

The **db2ckrst** utility can be used to query the database history and generate a list of backup image time stamps needed for an incremental restore. A simplified restore syntax for a manual incremental restore is also generated. It is recommended that you keep a complete record of backups, and use this utility only as a guide.

Limitations to automatic incremental restore

The automatic incremental restore is useful when you need to restore your database. However, you should consider the limitations of automatic incremental restore when you are deciding how you will recover your database to prevent unnecessary issues.

The following limitations affect automatic incremental restore:

1. If a table space name has been changed since the backup operation you want to restore from, and you use the new name when you issue a table space level restore operation, the required chain of backup images from the database history will not be generated correctly and an error will occur (SQL2571N).

   Example:

   ```
   db2 backup db sample —> <ts1>
   db2 backup db sample incremental —> <ts2>
   db2 rename tablespace from userspace1 to t1
   db2 restore db sample tablespace ('t1') incremental automatic taken
   at <ts2>

   SQL2571N Automatic incremental restore is unable to proceed.
   Reason code: "3".
   ```

   Suggested workaround: Use manual incremental restore.
2. If you drop a database, the database history will be deleted. If you restore the dropped database, the database history will be restored to its state at the time of the restored backup and all history entries after that time will be lost. If you then attempt to perform an automatic incremental restore that would need to use any of these lost history entries, the RESTORE utility will attempt to restore an incorrect chain of backups and will return an "out of sequence" error (SQL2572N).

   Example:

   ```
   db2 backup db sample —> <ts1>
   db2 backup db sample incremental —> <ts2>
   db2 backup db sample incremental delta —> <ts3>
   db2 backup db sample incremental delta —> <ts4>
   ```

```
db2 drop db sample
db2 restore db sample incremental automatic taken at <ts2>
db2 restore db sample incremental automatic taken at <ts4>
```

Suggested workarounds:

- Use manual incremental restore.
- Restore the history file first from image <ts4> before issuing an automatic incremental restore.

3. If you restore a backup image from one database into another database and then do an incremental (delta) backup, you can no longer use automatic incremental restore to restore this backup image.

   Example:

```
db2 create db a
db2 create db b

db2 update db cfg for a using trackmod on

db2 backup db a -> ts1
db2 restore db a taken at ts1 into b

db2 backup db b incremental -> ts2

db2 restore db b incremental automatic taken at ts2

SQL2542N No match for a database image file was found based on the source
database alias "B" and timestamp "ts1" provided.
```

   Suggested workaround:

   - Use manual incremental restore as follows:

```
db2 restore db b incremental taken at ts2
db2 restore db a incremental taken at ts1 into b
db2 restore db b incremental taken at ts2
```

   - After the manual restore operation into database B, issue a full database backup to start a new incremental chain

Chapter 45. BACKUP DATABASE command

Use the BACKUP DATABASE command to create a backup of your DB2 database and related stored data and to prevent data loss in the event of a database service outage.

The simplest form of the DB2 **BACKUP DATABASE** command requires only that you specify the alias name of the database that you want to back up. For example:

```
db2 backup db sample
```

In IBM Data Studio Version 3.1 or later, you can use the task assistant for backing up databases. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

If the command completes successfully, you will have acquired a new backup image that is located in the path or the directory from which the command was issued. It is located in this directory because the command in this example does not explicitly specify a target location for the backup image. Backup images created by DB2 Version 9.5 and later are generated with file mode 600, meaning that on UNIX only the instance owner has read and write privileges and on Windows only members of the DB2ADMNS (and Administrators) group have access to the backup images.

Note: If the DB2 client and server are not located on the same system, DB2 database systems will determine which directory is the current working directory on the client machine and use that as the backup target directory on the server. For this reason, it is recommended that you specify a target directory for the backup image.

Backup images are created at the target location specified when you invoke the backup utility. This location can be:

- A directory (for backups to disk or diskette)
- A device (for backups to tape)
- A Tivoli Storage Manager (TSM) server
- Another vendor's server

The recovery history file is updated automatically with summary information whenever you invoke a database backup operation. This file is created in the same directory as the database configuration file.

If you want to delete old backup images that are no longer required, you can remove the files if the backups are stored as files. If you subsequently run a **LIST HISTORY** command with the **BACKUP** option, information about the deleted backup images will also be returned. You must use the **PRUNE** command to remove those entries from the recovery history file.

If your recovery objects were saved using Tivoli Storage Manager (TSM), you can use the **db2adutl** utility to query, extract, verify, and delete the recovery objects. On

Linux and UNIX, this utility is located in the sqllib/adsm directory, and on Windows operating systems, it is located in sqllib\bin. For snapshots, use the **db2acsutil** utility located in sqllib/bin.

On all operating systems, file names for backup images created on disk consist of a concatenation of several elements, separated by periods:

```
DB_alias.Type.Inst_name.DBPARTnnn.timestamp.Seq_num
```

For example:

```
STAFF.0.DB201.DBPART000.19950922120112.001
```

Database alias
A 1- to 8-character database alias name that was specified when the backup utility was invoked.

Type Type of backup operation, where: 0 represents a full database-level backup, 3 represents a table space-level backup, and 4 represents a backup image generated by the **LOAD COPY TO** command.

Instance name
A 1- to 8-character name of the current instance that is taken from the **DB2INSTANCE** environment variable.

Database partition number
In single partition database environments, this is always DBPART000. In partitioned database environments, it is DBPART*xxx*, where *xxx* is the number assigned to the database partition in the db2nodes.cfg file.

Time stamp
A 14-character representation of the date and time at which the backup operation was performed. The time stamp is in the form *yyyymmddhhnnss*, where:

- *yyyy* represents the year (1995 to 9999)
- *mm* represents the month (01 to 12)
- *dd* represents the day of the month (01 to 31)
- *hh* represents the hour (00 to 23)
- *nn* represents the minutes (00 to 59)
- *ss* represents the seconds (00 to 59)

Sequence number
A 3-digit number used as a file extension.

When a backup image is written to tape:

- File names are not created, but the information described previously is stored in the backup header for verification purposes.
- A tape device must be available through the standard operating system interface. In a large partitioned database environment, however, it might not be practical to have a tape device dedicated to each database partition server. You can connect the tape devices to one or more TSM servers, so that access to these tape devices is provided to each database partition server.
- In a partitioned database environment, you can also use products that provide virtual tape device functions, such as REELlibrarian 4.2 or CLIO/S. You can use these products to access the tape device connected to other nodes (database partition servers) through a pseudo tape device. Access to the remote tape device is provided transparently, and the pseudo tape device can be accessed through the standard operating system interface.

You cannot back up a database that is not in a normal or backup-pending state. A table space that is in a normal or backup-pending state can be backed up. If the table space is not in a normal or backup-pending state, a backup may or may not be permitted.

Concurrent backup operations on the same table space are not permitted. Once a backup operation has been initiated on a table space, any subsequent attempts will fail (SQL2048N).

If a database or a table space is in a partially restored state because a system crash occurred during the restore operation, you must successfully restore the database or the table space before you can back it up.

A backup operation will fail if a list of the table spaces to be backed up contains the name of a temporary table space.

The backup utility provides concurrency control for multiple processes that are making backup copies of different databases. This concurrency control keeps the backup target devices open until all the backup operations have ended. If an error occurs during a backup operation, and an open container cannot be closed, other backup operations targeting the same drive might receive access errors. To correct such access errors, you must terminate the backup operation that caused the error and disconnect from the target device. If you are using the backup utility for concurrent backup operations to tape, ensure that the processes do not target the same tape.

Displaying backup information

You can use **db2ckbkp** to display information about existing backup images. This utility allows you to:

- Test the integrity of a backup image and determine whether or not it can be restored.
- Display information that is stored in the backup header.
- Display information about the objects and the log file header in the backup image.

Privileges, authorities, and authorization required to use backup

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the backup utility.

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects.

Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

Backing up data

Use the **BACKUP DATABASE** command to take a copy of the database data and store it on a different medium. This backup data can then be used in the case of a failure or damage to the original data.

You can back up an entire database, database partition, or only selected table spaces.

Before you begin

You do not need to be connected to the database that is to be backed up: the backup database utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the backup operation. If you are connected to a database that is to be backed up, you will be disconnected when the **BACKUP DATABASE** command is issued and the backup operation will proceed.

The database can be local or remote. The backup image remains on the database server, unless you are using a storage management product such as Tivoli Storage Manager (TSM) or DB2 Advanced Copy Services (ACS).

If you are performing an offline backup and if you have activated the database by using the **ACTIVATE DATABASE** command, you must deactivate the database before you run the offline backup. If there are active connections to the database, in order to deactivate the database successfully, a user with SYSADM authority must connect to the database, and issue the following commands:

```
CONNECT TO database-alias
QUIESCE DATABASE IMMEDIATE FORCE CONNECTIONS;
UNQUIESCE DATABASE;
TERMINATE;
DEACTIVATE DATABASE database-alias
```

In a partitioned database environment, you can use the **BACKUP DATABASE** command to back up database partitions individually, use the **ON DBPARTITIONNUM** command parameter to back up several of the database partitions at once, or use the **ALL DBPARTITIONNUMS** parameter to back up all of the database partitions simultaneously. You can use the **LIST DBPARTITIONNUMS** command to identify the database partitions that have user tables on them that you might want to back up.

Unless you are using a single system view (SSV) backup, if you are performing an *offline* backup in a partitioned database environment, you should back up the catalog partition separately from all other database partitions. For example, you can back up the catalog partition first, then back up the other database partitions. This action is necessary because the backup operation might require an exclusive database connection on the catalog partition, during which the other database partitions cannot connect. If you are performing an *online* backup, all database partitions (including the catalog partition) can be backed up simultaneously or in any order.

On a distributed request system, backup operations apply to the distributed request database and the metadata stored in the database catalog (wrappers, servers, nicknames, and so on). Data source objects (tables and views) are not backed up, unless they are stored in the distributed request database

If a database was created with a previous release of the database manager, and the database has not been upgraded, you must upgrade the database before you can back it up.

Restrictions

The following restrictions apply to the backup utility:

- A table space backup operation and a table space restore operation cannot be run at the same time, even if different table spaces are involved.
- If you want to be able to do rollforward recovery in a partitioned database environment, you must regularly back up the database on the list of nodes. You must also have at least one backup image of the rest of the nodes in the system (even those nodes that do not contain user data for that database). Two situations require the backed-up image of a database partition at a database partition server that does not contain user data for the database:
  – You added a database partition server to the database system after taking the last backup, and you need to do forward recovery on this database partition server.
  – Point-in-time recovery is used, which requires that all database partitions in the system are in rollforward pending state.
- Online backup operations for DMS table spaces are incompatible with the following operations:
  – load
  – reorganization (online and offline)
  – drop table space
  – table truncation
  – index creation
  – not logged initially (used with the CREATE TABLE and ALTER TABLE statements)
- If you attempt to perform an offline backup of a database that is currently active, you will receive an error. Before you run an offline backup, you can make sure that the database is not active by issuing the **DEACTIVATE DATABASE** command.

Procedure

To invoke the backup utility:
- Issue the **BACKUP DATABASE** command in the command line processor (CLP).
- Run the ADMIN_CMD procedure with the `BACKUP DATABASE` parameter.
- Use the db2Backup application programming interface (API).
- Open the task assistant in IBM Data Studio for the **BACKUP DATABASE** command.

Example

Following is an example of the **BACKUP DATABASE** command issued through the CLP:

```
db2 backup database sample to c:\DB2Backups
```

What to do next

If you performed an offline backup, after the backup completes, you must reactivate the database:

```
ACTIVATE DATABASE sample
```

Performing a snapshot backup

A snapshot backup operation uses the fast copying technology of a storage device to perform the data copying portion of the backup.

Before you begin

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, refer to this tech note.

Before you can perform a snapshot backup, you must enable DB2 Advanced Copy Services (ACS).

Restrictions

You cannot recover individual table spaces by using snapshot backups.

If you use integrated snapshot backups, you cannot perform a redirected restore. A FlashCopy® restore reverts the complete set of volume groups containing all database paths to a prior point in time.

Procedure

To perform a snapshot backup, use one of the following approaches:

- Issue the **BACKUP DATABASE** command with the **USE SNAPSHOT** parameter, as shown in the following example:

  ```
  db2 backup db sample use snapshot
  ```

- Call the ADMIN_CMD procedure with **BACKUP DB** and **USE SNAPSHOT** parameters, as shown in the following example:

  ```
  CALL SYSPROC.ADMIN_CMD
     ('backup db sample use snapshot')
  ```

- Issue the db2Backup API with the `SQLU_SNAPSHOT_MEDIA` media type, as shown in the following example:

  ```
  int sampleBackupFunction( char dbAlias[],
                            char user[],
                            char pswd[],
                            char workingPath[] )
  {
    db2MediaListStruct mediaListStruct = { 0 };

    mediaListStruct.locations = &workingPath;
    mediaListStruct.numLocations = 1;
    mediaListStruct.locationType = SQLU_SNAPSHOT_MEDIA;


    db2BackupStruct backupStruct = { 0 };

    backupStruct.piDBAlias = dbAlias;
    backupStruct.piUsername = user;
    backupStruct.piPassword = pswd;
    backupStruct.piVendorOptions = NULL;
    backupStruct.piMediaList = &mediaListStruct;

    db2Backup(db2Version950, &backupStruct, &sqlca);

    return 0;
  }
  ```

Using a split mirror as a backup image

Use the following procedure to create a split mirror of a database in a different location on the same system for use as a backup image outside of a DB2 pureScale environment. This procedure can be used instead of performing backup database operations on the database.

Procedure

To use a split mirror as a backup image:

1. Connect to the primary database by using the following command:

   ```
   db2 connect to db_name
   ```

2. Suspend the I/O write operations on the primary database by using the following command:

   ```
   db2 set write suspend for database
   ```

   While the database is in suspended state, you should not be running other utilities or tools. You should be only making a copy of the database. You can optionally flush all buffer pools before you issue **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved by using the FLUSH BUFFERPOOLS ALL statement.

3. Create one or multiple split mirrors from the primary database by using the appropriate operating system-level and storage-level commands.

   Note:

   - Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
   - If you specified `EXCLUDE LOGS` with the **SET WRITE** command, do not include the log files in the copy.

4. Resume the I/O write operations on the primary database by using the following command:

   ```
   db2 set write resume for database
   ```

   Assuming that a failure would occur on the system, perform the following steps to restore the database by using the split-mirror database as the backup:

   a. Stop the database instance by using the following command:

      ```
      db2stop
      ```

   b. Copy the split-off data by using operating system-level commands.

      Important: Do not copy the split-off log files, because the original logs are needed for rollforward recovery.

   c. Start the database instance by using the following command:

      ```
      db2start
      ```

   d. Initialize the primary database:

      ```
      db2inidb database_alias as mirror
      ```

      where *database_alias* represents the database alias.

   e. Roll forward the database to the end of the logs, or to a point-in-time, and stop.

Using a split mirror as a backup image in a DB2 pureScale environment

Use the following procedure to create a split mirror of a database in a different location on the same system for use as a backup image in a DB2 pureScale environment. This procedure can be used instead of performing backup database operations on the database.

Procedure

To use a split mirror as a backup image:

1. Connect to the primary database by using the following command:

   ```
   db2 connect to db_name
   ```

2. Configure the General Parallel File System (GPFS) on the secondary cluster by extracting and importing the settings from the primary cluster. On the primary cluster, run the following GPFS command:

   ```
   mmfsctl filesystem syncFSconfig -n remotenodefile
   ```

   where *remotenodefile* is the list of hosts in the secondary cluster.

3. Suspend the I/O write operations on the primary database by using the following command:

   ```
   db2 set write suspend for database
   ```

   While the database is in suspended state, you should not be running other utilities or tools. You should be only making a copy of the database. You can optionally flush all buffer pools before you issue **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved by using the FLUSH BUFFERPOOLS ALL statement.

4. Determine which file systems must be suspended and copied by using the following command:

   ```
   db2cluster -cfs -list -filesystem
   ```

5. Suspend each GPFS file system that contains container data or log data by using the following command:

   ```
   /usr/lpp/mmfs/bin/mmfsctl filesystem suspend-write
   ```

   where *filesystem* represents a file system that contains data or log data.

   Note: While the GPFS file systems are suspended, write operations are blocked. You should only be performing the split mirror operations during this period to minimize the amount of time that operations are blocked.

6. Create one or multiple split mirrors from the primary database by using the appropriate operating system-level and storage-level commands.

   Note:

   - Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
   - If you specified EXCLUDE LOGS with the **SET WRITE** command, do not include the log files in the copy.

7. Resume the GPFS file systems that were suspended by using the following command for each suspended file system:

   ```
   /usr/lpp/mmfs/bin/mmfsctl filesystem resume
   ```

where *filesystem* represents a suspended file system that contains data or log data.

8. Resume the I/O write operations on the primary database by using the following command:

   ```
   db2 set write resume for database
   ```

   Assuming that a situation requires you to restore the database by using the split mirror as the backup image, perform the following steps:

   a. Stop the primary database instance by using the following command:

      ```
      db2stop
      ```

   b. List the cluster manager domain by using the following command:

      ```
      db2cluster -cm -list -domain
      ```

   c. Stop the cluster manager on each host in the cluster by using the following command:

      ```
      db2cluster -cm -stop -host host -force
      ```

      Note: The last host which you shut down must be the host from which you are issuing this command.

   d. Stop the GPFS cluster on the primary database instance by using the following command:

      ```
      db2cluster -cfs -stop -all
      ```

   e. Copy the split-off data off the primary database by using appropriate operating system-level commands.

      Important: Do not copy the split-off log files, because the original logs are needed for rollforward recovery.

   f. Start the GPFS cluster on the primary database instance by using the following command:

      ```
      db2cluster -cfs -start -all
      ```

   g. Start the cluster manager by using the following command:

      ```
      db2cluster -cm -start -domain domain
      ```

   h. Start the database instance by using the following command:

      ```
      db2start
      ```

   i. Initialize the primary database by using the following command:

      ```
      db2inidb database_alias as mirror
      ```

   j. Roll forward the primary database to the end of the logs, or to a point-in-time, and stop.

Backing up to tape

When you back up your database or table space, you must correctly set your block size and your buffer size. This is particularly true if you are using a variable block size (on AIX, for example, if the block size has been set to zero).

There is a restriction on the number of fixed block sizes that can be used when backing up. This restriction exists because DB2 database systems write out the backup image header as a 4-KB block. The only fixed block sizes DB2 database systems support are 512, 1024, 2048, and 4096 bytes. If you are using a fixed block size, you can specify any backup buffer size. However, you might find that your backup operation will not complete successfully if the fixed block size is not one of the sizes that DB2 database systems support.

If your database is large, using a fixed block size means that your backup operations might take more time than expected to complete. To improve performance, you can use a variable block size.

Note: When using a variable block size, ensure that you have well tested procedures in place that enable you to recover successfully, including explicitly specified buffer sizes for the **BACKUP** and **RESTORE** commands, with backup images that are created using a variable block size.

When using a variable block size, you must specify a backup buffer size that is less than or equal to the maximum limit for the tape devices that you are using. For optimal performance, the buffer size must be equal to the maximum block size limit of the device being used.

Before a tape device can be used on a Windows operating system, the following command must be issued:

```
db2 initialize tape on device using blksize
```

Where:

device is a valid tape device name. The default on Windows operating systems is `\\.\TAPE0`.

blksize is the blocking factor for the tape. It must be a factor or multiple of 4096. The default value is the default block size for the device.

Restoring from a backup image with variable block size might return an error. If this happens, you might need to rewrite the image using an appropriate block size. Following is an example on AIX:

```
tctl -b 0 -Bn -f /dev/rmt0 read > backup_filename.file
dd if=backup_filename.file of=/dev/rmt0 obs=4096 conv=sync
```

The backup image is dumped to a file called `backup_filename.file`. The **dd** command dumps the image back onto tape, using a block size of 4096.

There is a problem with this approach if the image is too large to dump to a file. One possible solution is to use the **dd** command to dump the image from one tape device to another. This will work as long as the image does not span more than one tape. When using two tape devices, the **dd** command is:

```
dd if=/dev/rmt1 of=/dev/rmt0 obs=4096
```

If using two tape devices is not possible, you might be able to dump the image to a raw device using the **dd** command, and then to dump the image from the raw device to tape. The problem with this approach is that the **dd** command *must* keep track of the number of blocks dumped to the raw device. This number must be specified when the image is moved back to tape. If the **dd** command is used to dump the image from the raw device to tape, the command dumps the entire contents of the raw device to tape. The **dd** utility cannot determine how much of the raw device is used to hold the image.

When using the backup utility, you will need to know the maximum block size limit for your tape devices. Here are some examples:

Device	Attachment	Block Size Limit	DB2 Buffer Size Limit (in 4-KB pages)
8 mm	scsi	131,072	32
3420	s370	65,536	16
3480	s370	61 440	15
3490	s370	61 440	15
3490E	s370	65,536	16
7332 (4 mm)[1]	scsi	262,144	64
3490e	scsi	262,144	64
3590[2]	scsi	2,097,152	512
3570 (magstar MP)		262,144	64

Note:

1. The 7332 does not implement a block size limit. 256 KB is simply a suggested value. Block size limit is imposed by the parent adapter.
2. While the 3590 does support a 2-MB block size, you could experiment with lower values (like 256 KB), provided the performance is adequate for your needs.
3. For information about your device limit, check your device documentation or consult with the device vendor.

Verifying the compatibility of your tape device

On UNIX, Linux, and AIX operating systems only, to determine whether your tape device is supported for backing up your DB2 databases, perform the following procedure:

As the database manager instance owner, run the operating system command **dd** to read from or write to your tape device. If the **dd** command succeeds, then you can back up your DB2 databases using your tape device.

Backing up to named pipes

Support is now available for database backup to (and database restore from) local named pipes on UNIX operating systems.

Before you begin

Both the writer and the reader of the named pipe must be on the same machine. The pipe must exist on a local file system. Because the named pipe is treated as a local device, there is no need to specify that the target is a named pipe.

Procedure

1. Create a named pipe. The following is an AIX example:

   ```
   mkfifo /u/dmcinnis/mypipe
   ```

2. If this backup image is going to be used by the restore utility, the restore operation must be invoked *before* the backup operation, so that it does not miss any data:

   ```
   db2 restore db sample from /u/dmcinnis/mypipe into mynewdb
   ```

3. Use this pipe as the target for a database backup operation:

```
db2 backup db sample to /u/dmcinnis/mypipe
```

Backing up partitioned databases

Backing up a database in a partitioned database environment can pose difficulties such as tracking the success of the backup of each database partition, managing the multiple log files and backup images, and ensuring the log files and backup images for all the database partitions span the minimum recovery time that is required to restore the database.

Using a single system view (SSV) backup is the easiest way to back up a partitioned database.

About this task

There are four ways to back up a database in a partitioned database environment:

- Back up each database partition one at a time by using the **BACKUP DATABASE** command, the **BACKUP DATABASE** command with the ADMIN_CMD procedure, or the db2Backup API.
- Use the **db2_all** command with the **BACKUP DATABASE** command to first back up the catalog partition and then to back up a specified list of database partitions.
- Run a single system view (SSV) backup to back up some or all of the database partitions simultaneously, including the catalog partition.
- Use a task assistant in IBM Data Studio to guide you through the process of backing up the database.

Backing up each database partition one at a time is time-consuming and error-prone. Backing up all the partitions by using the **db2_all** command is easier than backing up each database partition individually because you generally only must make one command call. However, when you use **db2_all** to back up a partitioned database, you sometimes still must make multiple calls to **db2_all** because the database partition that contains the catalog cannot be backed up simultaneously with non-catalog database partitions. Whether you back up each database partition one at a time or use **db2_all**, managing backup images that were created using either of these methods is difficult because the time stamp for each database partition's backup image is different, and coordinating the minimum recovery time across the database partitions' backup images is difficult as well.

For the previously mentioned reasons, the recommended way to back up a database in a partitioned database environment is to use an SSV backup because you can decide to back up all database partitions simultaneously, including the catalog partition, and get the same time stamp for each database partition backup. Alternatively, you can split your backup, specifying some database partitions for which you get the same time stamp, and later take additional backups on the other database partitions to complete the database backup. The catalog partition can be backed up at any time with any other database partitions.

Note: For restore operations, you still must restore the catalog partition before you restore some or all of the other database partitions.

Procedure

To back up some or all of the database partitions of a partitioned database simultaneously by using an SSV backup:

1. Optional: Allow the database to remain online, or take the database offline.

You can back up a partitioned database while the database is online or offline. If the database is online, the backup utility acquires shared connections to the other database partitions, so user applications are able to connect to database partitions while they are being backed up.

2. On the database partition that contains the database catalog, perform the backup with appropriate parameters for partitioned databases, using one of the following methods:
   - Run the **BACKUP DATABASE** command with the **ON DBPARTITIONNUMS** parameter.
   - Run the **BACKUP DATABASE** command with the **ON DBPARTITIONNUMS** parameter by using the ADMIN_CMD procedure.
   - Call the db2Backup API with the **iAllNodeFlag** parameter.
   - Open the task assistant for the **BACKUP DATABASE** command in IBM Data Studio.
3. Optional: Include the log files that are required for recovery with the backup images.

   By default, log files are included with backup images if you are performing an SSV backup (that is, if you specify the **ON DBPARTITIONNUM** parameter). If you do not want log files to be included with the backup images, use the **EXCLUDE LOGS** command parameter when you run the backup. Log files are excluded from the backup image by default for non-SSV backups.

   For more information, see "Including log files with a backup image" on page 747.
4. Optional: Delete previous backup images. The method that you use to delete old backup images depends on how you store the backup images. For example, if you store the backup images to disk, you can delete the files; if you store the backup images using Tivoli Storage Manager, you can use the **db2adutl** utility to delete the backup images. If you are using DB2 Advanced Copy Services (ACS), you can use the **db2acsutil** to delete snapshot backup objects.

Backup and restore operations in a DB2 pureScale environment

In a DB2 pureScale environment, issuing a single **BACKUP DATABASE** or **RESTORE DATABASE** command on any member initiates a backup or restore operation on behalf of all members.

Because a DB2 pureScale environment can have only one database partition, a backup operation has only one set of data to process and produces only one backup image for the entire group. In the case of the other members, only the database metadata and transaction logs must be processed, and those are included in the single backup image.

A backup image includes data from the specified table spaces and any required metadata and configuration information for all currently defined members. You do not have to perform additional backup operations on any other member in the DB2 pureScale instance. Moreover, you require only a single **RESTORE DATABASE** command to restore the database and the member-specific metadata for all members. You do not have to perform additional restore operations on any other member to restore the cluster. The time stamps of consecutive backup images are unique, increasing values, regardless of which member produced them.

All members must be consistent before an offline backup operation can be attempted. Only one offline backup operation can run at one time, because the backup utility acquires super-exclusive access to the database across all members.

Although concurrent online backup operations are supported, different backup operations cannot copy the same table spaces simultaneously, and must wait their turn.

All of the reading of data and metadata from the database and all of the writing to a backup image takes place on a single member. Interactions between the backup or restore operation and other members are limited to copying or updating database metadata (such as table space definitions, the log file header, and the database configuration).

Note: Before taking a backup, you need to ensure that the log archiving path is set to a shared directory so that all the members are able to access the logs for subsequent rollforward operations. If the archive path is not accessible from the member on which the rollforward is being executed, SQL1273N is returned. The following command is an example of how to set the log path to the shared directory:

```
db2 update db cfg using logarchmeth1
     DISK:/db2fs/gpfs1/svtdbm5/svtdbm5/ArchiveLOGS
```

(where *gpfs1* is the shared directory for the members and *ArchiveLOGS* is the actual directory that archives the logs.

Online backup operations can proceed successfully if another member is offline, goes offline, or comes back online while the operation is executing (Table 124). Although database restore operations are not affected by the state of other members, backup operations might have to wait for a short duration while member crash recovery is completed on an offline and inconsistent member.

Table 124. Effect of the state of other members in a DB2 pureScale instance on database backup and restore operations

	State of other members	
Operation	**Offline and consistent**	**Offline and inconsistent**
Online backup	The backup operation succeeds. The other member cannot become active while the backup utility is accessing the log file header (LFH) near the beginning of the backup operation or while the backup utility is accessing the log stream near the end of the backup operation.	The backup operation succeeds, but it must wait for member crash recovery to be completed and for the other member to become either active or consistent. The other member cannot become active while the backup utility is accessing the LFH near the beginning of the backup operation or while the backup utility is accessing the log stream near the end of the backup operation.
Restore	The restore operation is completed normally.	The restore operation is completed normally.

Image and archive naming

File names for backup images that you create on disk consist of a concatenation of several elements, separated by periods:

```
DB_alias.Type.Inst_name.DBPARTnnn.Timestamp.Seq_num
```

DB_alias

The database alias name that you specified when you invoked the backup utility.

Type The type of backup operation, where 0 represents a full database backup, 3 represents a table space backup, and 4 represents a backup image generated by the **LOAD** command with the **COPY** NO option.

Inst_name

The name of the current instance, which is the value of the **DB2INSTANCE** environment variable.

nnn The database partition number. In a DB2 pureScale environment, the number is always 000.

Timestamp

A 14-character representation of the date and time when you performed the backup operation. The time stamp is in the form *yyyymmddhhnnss*, where:

- *yyyy* represents the year.
- *mm* represents the month (01 to 12).
- *dd* represents the day of the month (01 to 31).
- *hh* represents the hour (00 to 23).
- *nn* represents the minutes (00 to 59).
- *ss* represents the seconds (00 to 59).

Seq_num

A 3-digit number used as a file extension.

For example:

```
SAMPLE.0.krodger.DBPART000.200802241234.001
```

Online backup with INCLUDE LOGS

An online backup operation with the **INCLUDE LOGS** option (the default) produces a backup image that includes the range of log files required to restore and roll the database forward to its minimum recovery time. If this backup image is then used to restore to a new database (perhaps during disaster recovery), and only the logs from the backup image are available during a subsequent roll-forward operation, a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** parameter often returns an error message about a missing log file (SQL1273N). This is expected in some situations, because the database manager might have detected that additional logs were written after the backup operation, but that those logs are not available for the current roll-forward operation. It might also be the case that one or more of the logs that are necessary to roll the database forward to a consistent point in time are missing. In either case, verify that the end point of the roll-forward operation is acceptable and then issue a **ROLLFORWARD DATABASE** with the **AND STOP** parameter. If the roll-forward operation has reached its minimum recovery time despite the missing log file, the **ROLLFORWARD DATABASE** with the **AND STOP** parameter should complete successfully; otherwise, it returns SQL1276N (the roll-forward operation did not reach its minimum recovery time using this backup image).

Disaster recovery and high availability through log shipping in a DB2 pureScale environment

Log shipping is the process of copying whole log files to a standby machine, either from an archive device, or through a user exit program running against the primary database. You can choose to keep a standby database up-to-date by applying the logs to it as they are archived, or you can keep the database or table space backup images and log archives on the standby site, and perform restore and roll-forward operations only after a disaster has occurred. In either case, the roll-forward operation on the standby site might detect that one or more log files are missing and return SQL1273N. Verify that the roll-forward operation reached an acceptable time stamp, or take appropriate action to correct the problem.

If, during a log stream merge operation, the DB2 database manager determines that there is a missing log file in one of the log streams, an error is returned. The roll-forward utility returns SQL1273N; the db2ReadLog API returns SQL2657N. If you choose to keep a standby database up-to-date by applying logs to it as they are archived, roll-forward operations might frequently detect that some logs are missing.

Figure 58 shows an example of how two members could write log records to the log files in their active log stream. Each log file is represented by a box. Consider a scenario where both a primary and standby site have been set up for high availability. A **ROLLFORWARD DATABASE** command with the **END OF LOGS** option is attempted on the standby site at time points A, B and C. For any particular point in time, any log files that have been closed before that time have been archived and are accessible on the standby. Otherwise, the log file is still active on the primary and is not available to the standby yet (as shown for log file 4 on log stream 1 at time B).

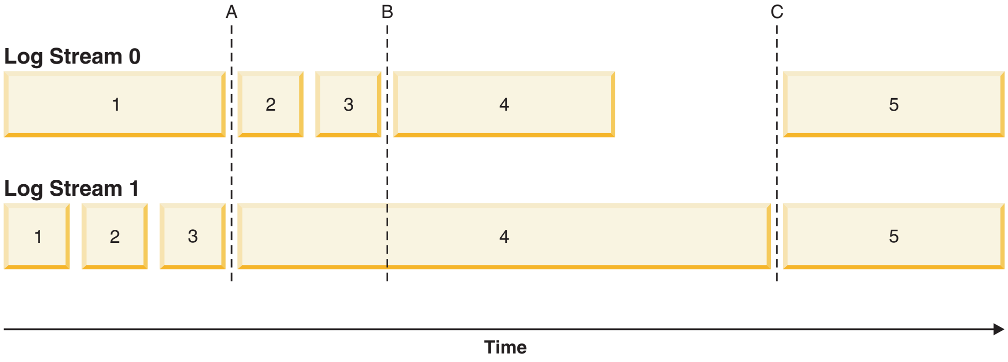


Figure 58. Log files in a DB2 pureScale environment

At time A, the **ROLLFORWARD DATABASE** command will complete successfully as log file 1 from log stream 0 was closed and archived at the same time as log file 3 from log stream 1. At time B however, the **ROLLFORWARD DATABASE** command will return v. This happens because at the time that the command is issued on the standby site, the standby site has access to log files 2 and 3 from log stream 0, but not to log file 4 from log stream 1 because the log file is still open and active on the primary site. Furthermore, since the log records in files 2 and 3 on log stream 0 were written during the same time period as the beginning of log file 4 on log stream 1, the roll-forward operation cannot process log files 2 and 3 until log file 4 from log stream 1 is made available. At time C, when log file 4 is finally closed and archived on log stream 1, a **ROLLFORWARD DATABASE** command will complete successfully. It is possible to force the truncation and archiving of files across all the log streams using the **ARCHIVE LOG** command, or by deactivating the database across all members. In the case of the **ARCHIVE LOG** command, the current log file on each log stream is truncated independently and there is no guarantee that it will happen at the exact same point in time across all members. Therefore, even if the **ARCHIVE LOG** command is issued, it is still possible to get an SQL1273N error when executing the **ROLLFORWARD DATABASE** command.

While missing log conditions are common and expected when using log shipping in a DB2 pureScale environment, in most cases, each roll-forward operation on the standby will make additional progress over the last **ROLLFORWARD DATABASE** command (even when SQL1273N is returned) and therefore the error itself should often be expected. It is possible, however, for the primary site to have trouble archiving a file for one log stream while successfully archiving logs for the other log streams. This could be the result of a temporary problem accessing the archive storage for one log stream. Such problems can cause the log merge and replay on the standby to be held up, increasing the number of transactions that could be lost in the event of a disaster. To ensure that your standby system is up-to-date, issue a **ROLLFORWARD DATABASE** command with the **QUERY STATUS** parameter after each roll-forward operation that returns SQL1273N and verify that progress is being made over time. If a roll-forward operation on the standby is not making progress over an extended period of time, determine why the log file reported as missing is not available on the standby system and correct the problem. The **ARCHIVE LOG** command can be used to truncate the log files that are currently being updated on each member, making them eligible for archiving and subsequent replay on the standby system.

In the event of a disaster (for example, fire, earthquake, vandalism, or other catastrophic events) your plan for recovery might be to execute a roll-forward operation through all remaining logs, or a restore and roll-forward operation through all available logs. As mentioned previously, the roll-forward operation might detect that one or more log file is missing, because log files were written on the primary but not yet archived at the time of the disaster (SQL1273N). It is also possible that a log that was archived cannot be found by the roll-forward utility for some unexpected reason; this can also cause the roll-forward utility to return SQL1273N. It is important to validate the end point of a roll-forward operation by using the **ROLLFORWARD DATABASE** command with the **QUERY STATUS** parameter, and to decide whether or not the missing log condition is expected. If the missing log condition is expected, or the end point is acceptable, you can issue a **ROLLFORWARD DATABASE** command with the **STOP** parameter to complete the roll-forward recovery process.

Restrictions

Backup and restore operations between an environment where the DB2 pureScale Feature is installed and an environment where the DB2 pureScale Feature is not installed are not supported.

After a change in topology that involves adding or dropping a member, you cannot perform roll-forward recovery operations through the point where the topology change occurred. If you add or drop a member, the database is placed in backup pending state, and you must perform a full database backup operation before a connection to the database can be made. To recover, restore this backup image and roll forward to the end of the logs. If you must restore a backup image from before the topology change, you will only be able to roll forward to the point at which the topology change occurred. This can be accomplished by issuing a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** parameter (which returns SQL1546N) followed by a **ROLLFORWARD DATABASE** command with the **STOP** parameter. This operation will not recover any transactions that changed the database after the topology change.

In a DB2 pureScale environment, the **ON ALL DBPARTITIONNUMS** parameter and the **ON DBPARTITION (0)** parameter of the **BACKUP DATABASE** command are valid. If you specify a database partition number other than 0, however, an error (SQL0270N) is returned because no other database partitions exist.

The following restriction applies to this release:

- A database, which resides outside of a DB2 pureScale environment, can be migrated to a DB2 pureScale environment. You cannot use database restore operations to migrate such database to a DB2 pureScale environment.
- Delta and incremental backup operations are not supported.
- In Version 10 GA and Fix Pack 1, snapshot backup operations using DB2 Advanced Copy Services (ACS) are not supported. In Version 10 Fix Pack 2, this restriction is removed.

Examples

- Back up a 4-member database named SAMPLE from any member:
  ```
  BACKUP DB SAMPLE
  ```
- Restore a 1-member database named SAMPLE:
  ```
  RESTORE DB SAMPLE
  ```
- Use the **RECOVER DATABASE** command to restore and roll forward a database named SAMPLE from any member:
  ```
  RECOVER DB SAMPLE TO END OF LOGS
  ```

  If the database does not exist, use the **RESTORE DATABASE** and **ROLLFORWARD DATABASE** commands instead of the **RECOVER DATABASE** command because an existing database with a complete database history is required for the successful completion of the **RECOVER DATABASE** command.

Enabling automatic backup

A database can become unusable due to a wide variety of hardware or software failures. Ensuring that you have a recent, full backup of your database is an integral part of planning and implementing a disaster recovery strategy for your system.

Use automatic database backup as part of your disaster recovery strategy to enable DB2 to back up your database both properly and regularly.

About this task

You can configure automatic backup using the command line interface, or the AUTOMAINT_SET_POLICY system stored procedure. You also need to enable the health indicator db.db_backup_req, which by default is enabled. Note that only an active database is considered for the evaluation.

Procedure

- To configure automatic backup using the command line interface, set each of the following database configuration parameters to `ON`:
  - **AUTO_MAINT**
  - **AUTO_DB_BACKUP**
- To configure automatic backup usingIBM Data Studio, right-click the database and select the task assistant to configure automatic backup.

- To configure automatic backup using the AUTOMAINT_SET_POLICY system stored procedure:
  1. Create configuration XML input specifying details like backup media, whether the backup should be online or offline, and frequency of the backup.

     You can copy the contents of the sample file called `DB2DefaultAutoBackupPolicy.xml` in the `SQLLIB/samples/automaintcfg` directory and modify the XML to satisfy your configuration requirements.
  2. Optional: Create an XML input file containing your configuration XML input.
  3. Call AUTOMAINT_SET_POLICY with the following parameters:
     - maintenance type: AutoBackup
     - configuration XML input: either a BLOB containing your configuration XML input text; or the name of the file containing your configuration XML input.

     See the topic "Configuring an automated maintenance policy using SYSPROC.AUTOMAINT_SET_POLICY or SYSPROC.AUTOMAINT_SET_POLICYFILE" for more information about using the AUTOMAINT_SET_POLICY system stored procedure.

Configuring an automated maintenance policy using SYSPROC.AUTOMAINT_SET_POLICY or SYSPROC.AUTOMAINT_SET_POLICYFILE

You can use the system stored procedures AUTOMAINT_SET_POLICY and AUTOMAINT_SET_POLICYFILE to configure the automated maintenance policy for a database.

Procedure

To configure the automated maintenance policy for a database, perform the following steps:

1. Connect to the database
2. Call AUTOMAINT_SET_POLICY or AUTOMAINT_SET_POLICYFILE
   - The parameters required for AUTOMAINT_SET_POLICY are:
     a. Maintenance type, specifying the type of automated maintenance activity to configure.
     b. Pointer to a BLOB that specifies the automated maintenance policy in XML format.
   - The parameters required for AUTOMAINT_SET_POLICYFILE are:
     a. Maintenance type, specifying the type of automated maintenance activity to configure.
     b. The name of an XML file that specifies the automated maintenance policy.

   Valid maintenance type values are:
   - AUTO_BACKUP - automatic backup
   - AUTO_REORG - automatic table and index reorganization
   - AUTO_RUNSTATS - automatic table **RUNSTATS** operations
   - MAINTENANCE_WINDOW - maintenance window

What to do next

You can use the system stored procedures AUTOMAINT_GET_POLICY and AUTOMAINT_GET_POLICYFILE to retrieve the automated maintenance policy configured for a database.

Monitoring backup operations

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

Procedure

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

```
list utilities show detail
```

Results

For backup operations, an initial estimate of the number of bytes to be processed will be specified. As the backup operation progresses the number of bytes to be processed will be updated. The bytes shown does not correspond to the size of the image and should not be used as an estimate for backup image size. The actual image might be much smaller depending on whether it is an incremental or compressed backup.

Example

The following is an example of the output for monitoring the performance of an offline database backup operation:

```
ID                                = 3
Type                              = BACKUP
Database Name                     = SAMPLE
Partition Number                  = 0
Description                       = offline db
Start Time                        = 08/04/2011 12:16:23.248367
State                             = Executing
Invocation Type                   = User
Throttling:
   Priority                       = Unthrottled
Progress Monitoring:
   Extimated Percentage Complete = 31
      Total Work                  = 123147277 bytes
      Completed Work              = 37857269 bytes
      Start Time                  = 08/04/2011 12:16:23.248377
```

Optimizing backup performance

When you perform a backup operation, the DB2 database manager automatically chooses an optimal value for the number of buffers, the buffer size, and the parallelism settings. The values are based on the amount of utility heap memory available, the number of processors available, and the database configuration.

Therefore, depending on the amount of storage available on your system, consider allocating more memory by increasing the **util_heap_sz** configuration parameter.

The objective is to minimize the time it takes to complete a backup operation. Unless you explicitly enter a value for the following **BACKUP DATABASE** command parameters, the DB2 database manager selects one for them:

- **WITH** *num-buffers* **BUFFERS**
- **PARALLELISM** *n*
- **BUFFER** *buffer-size*

If the number of buffers and the buffer size are not specified, resulting in the DB2 database manager setting the values, it should have minimal effect on large databases. However, for small databases, it can cause a large percentage increase in backup image size. Even if the last data buffer written to disk contains little data, the full buffer is written to the image anyway. In a small database, this means that a considerable percentage of the image size might be empty.

You can also choose to do any of the following to reduce the amount of time required to complete a backup operation:

- Specify table space backup.

  You can back up (and subsequently recover) part of a database by using the **TABLESPACE** option on the **BACKUP DATABASE** command. This facilitates the management of table data, indexes, and long field or large object (LOB) data in separate table spaces.
- Increase the value of the **PARALLELISM** parameter on the **BACKUP DATABASE** command so that it reflects the number of table spaces being backed up.

  The **PARALLELISM** parameter defines the number of processes or threads that are started to read data from the database and to compress data during a compressed backup operation. Each process or thread is assigned to a specific table space, so there is no benefit to specifying a value for the **PARALLELISM** parameter that is larger than the number of table spaces being backed up. When it finishes backing up this table space, it requests another. Note, however, that each process or thread requires both memory and CPU overhead.
- Increase the backup buffer size.

  The ideal backup buffer size is a multiple of the table space extent size plus one page. If you have multiple table spaces with different extent sizes, specify a value that is a common multiple of the extent sizes plus one page.
- Increase the number of buffers.

  Use at least twice as many buffers as backup targets (or sessions) to ensure that the backup target devices do not have to wait for data.
- Use multiple target devices.

Compatibility of online backup and other utilities

Some utilities can be run at the same time as an online backup, but others cannot.

The following utilities are compatible with online backup:

- **EXPORT**
- **INSPECT**

The following SQL statements and utilities are compatible with online backup only under certain circumstances:

- CREATE INDEX

  In SMS mode, online index create and online backup do not run concurrently due to the ALTER TABLE lock. Online index create acquires it in exclusive mode while online backup acquires it in share.

  In DMS mode, online index create and online backup can run concurrently in most cases. There is a possibility if you have a large number of tables in the same tablespace as the one in which you are creating the index, that the online index create will internally acquire an online backup lock that will conflict with any concurrent online backup.

- REORG INDEX with the ONLINE option

  As with online index create, in SMS mode, online index reorganization do not run concurrently with online backup due to the ALTER TABLE lock. Online index reorganization acquires it in exclusive mode while online backup acquires it in share. In addition, an online index reorganization operation, quiesces the table before the switch phase and acquires a Z lock, which prevents an online backup. However, the ALTER TABLE lock should prevent an online backup from running concurrently before the Z table lock is acquired.

  In DMS mode, online index reorganization and online backup can run concurrently.

  In addition, online index reorganization quiesces the table before the switch phase and gets a Z lock, which prevents an online backup.

- **IMPORT**

  The import utility is compatible with online backup except when the **IMPORT** command is issued with the **REPLACE** parameter, in which case, import gets a Z lock on the table and prevents an online backup from running concurrently.

- TRUNCATE TABLE

  The TRUNCATE statement is not compatible with online backup because it gets a Z lock on the table and prevents an online backup from running concurrently.

- **ALLOW READ ACCESS LOAD**

  ALLOW READ ACCESS load operations are not compatible with online backup when the **LOAD** command is issued with the **COPY NO** parameter. In this mode the utilities both modify the table space state, causing one of the utilities to report an error.

  ALLOW READ ACCESS load operations are compatible with online backup when the **LOAD** command is issued with the **COPY YES** option, although there might still be some compatibility issues. In SMS mode, the utilities can execute concurrently, but they will hold incompatible table lock modes and consequently might be subject to table lock waits. In DMS mode, the utilities both hold incompatible "Internal-B" (OLB) lock modes and might be subject to waits on that lock. If the utilities execute on the same table space concurrently, the load utility might be forced to wait for the backup utility to complete processing of the table space before the load utility can proceed.

- REORG TABLE with the ONLINE option

  The cleanup phase of online table reorganization cannot start while an online backup is running. You can pause the table reorganization, if required, to allow the online backup to finish before resuming the online table reorganization.

  You can start an online backup of a DMS table space when a table within the same table space is being reorganized online. There might be lock waits associated with the reorganization operation during the truncate phase.

  You cannot start an online backup of an SMS table space when a table within the same table space is being reorganized online. Both operations require an exclusive lock.

- DDLs that require a Z lock (such as ALTER TABLE, DROP TABLE, and DROP INDEX)

  Online DMS table space backup is compatible with DDLs that require a Z lock.

  Online SMS table space backup must wait for the Z lock to be released.

- Storage group DDLs

If you are modifying the database storage groups by issuing one of the following statements, you should take care to coordinate this operation with your online backup schedule:

  - CREATE STOGROUP
  - ALTER STOGROUP
  - DROP STOGROUP
  - RENAME STOGROUP
  - ALTER DATABASE

  If there is an online backup in progress, the storage group DDL waits behind that operation until it can obtain the appropriate lock, which can potentially take a long time. Similarly, an online backup waits behind any in-progress storage group DDL, until that DDL is committed or rolled back.

- **RUNSTATS** with the ALLOW WRITE or ALLOW READ option

  The **RUNSTATS** command is compatible with online backup except when the system catalog table space is an SMS table space. If the system catalog resides in an SMS table space, then the **RUNSTATS** command and the online backup hold incompatible table locks on the table causing lock waits.
- ALTER TABLESPACE

  Operations that enable or disable autoresize, or alter autoresize containers, are not permitted during an online backup of a table space.
- ALTER TABLESPACE with the REBALANCE option

  When online backup and rebalancer are running concurrently, online backup pauses the rebalancer and does not wait for it to complete.

The following utilities are not compatible with online backup:

- **REORG TABLE**
- **RESTORE DATABASE**
- **ROLLFORWARD DATABASE**
- **LOAD** with the **ALLOW NO ACCESS** option
- **SET WRITE**
- **BACKUP DATABASE** with the ONLINE option

  This applies to database-level online backups and table-space-level online backups (if they involve the same table space or table spaces).

Chapter 46. RECOVER DATABASE command

Use the RECOVER DATABASE command to perform the necessary restore and rollforward operations to recover a database to a specified time, which is based on information that is found in the recovery history file.

When you use this utility, you specify that the database be recovered to a point-in-time or to the end of the log files. The utility will then select the best suitable backup image and perform the recovery operations.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for recovering databases. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

The recover utility does not support the following **RESTORE DATABASE** command options:

- **TABLESPACE** *tablespace-name*. Table space restore operations are not supported.
- **INCREMENTAL**. Incremental restore operations are not supported.
- **OPEN** *num-sessions* **SESSIONS**. You cannot indicate the number of I/O sessions that are to be used with TSM or another vendor product.
- **BUFFER** *buffer-size*. You cannot set the size of the buffer used for the restore operation.
- **DLREPORT** *filename*. You cannot specify a file name for reporting files that become unlinked.
- **WITHOUT ROLLING FORWARD**. You cannot specify that the database is not to be placed in rollforward pending state after a successful restore operation.
- **PARALLELISM** *n*. You cannot indicate the degree of parallelism for the restore operation.
- **WITHOUT PROMPTING**. You cannot specify that a restore operation is to run unattended

In addition, the recover utility does not allow you to specify any of the **REBUILD** options. However, the recovery utility will automatically use the appropriate **REBUILD** option if it cannot locate any database backup images based on the information in the recovery history file.

For the **RECOVER DATABASE** command, you cannot use the `TABLESPACE` option or the `INCREMENTAL` option from the **RESTORE DATABASE** command.

For the **RECOVER DATABASE** command, `restore` option is automated. Same applies for the `REBUILD` option in the **RESTORE** command.

Privileges, authorities, and authorization required to use recover

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the recover utility.

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects.

Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

Recovering data

The **RECOVER DATABASE** command recovers a database and all storage groups to a specified time, by using information found in the recovery history file.

Before you begin

If you issue the **RECOVER DATABASE** command following an incomplete recover operation that ended during the rollforward phase, the recover utility attempts to continue the previous recover operation, without redoing the restore phase. If you want to force the recover utility to redo the restore phase, issue the **RECOVER DATABASE** command with the **RESTART** option to force the recover utility to ignore any prior recover operation that failed to complete. If you are using the application programming interface (API), specify the caller action DB2RECOVER_RESTART for the **iRecoverAction** field to force the recover utility to redo the restore phase.

If the **RECOVER DATABASE** command is interrupted during the restore phase, it cannot be continued. You must reissue the **RECOVER DATABASE** command.

You should not be connected to the database that is to be recovered: the recover database utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the recover operation.

About this task

The database can be local or remote.

Note: In a partitioned database environment, the recover utility must be invoked from the catalog partition of the database.

Procedure

To invoke the recover utility, use the:

- **RECOVER DATABASE** command, or
- db2Recover application programming interface (API).

Example

The following example shows how to use the **RECOVER DATABASE** command through the CLP:

```
db2 recover db sample
```

Optimizing recovery performance

There are strategies that you can use to improve DB2 performance during database recovery and decrease the time that is required to recover from a DB2 service outage.

The following should be considered when thinking about recovery performance:

- You can improve performance for databases that are frequently updated by placing the logs on a separate device. In the case of an online transaction processing (OLTP) environment, often more I/O is needed to write data to the logs than to store a row of data. Placing the logs on a separate device will minimize the disk arm movement that is required to move between a log and the database files.

  You should also consider what other files are on the disk. For example, moving the logs to the disk used for system paging in a system that has insufficient real memory will defeat your tuning efforts.

  DB2 database products automatically attempt to minimize the time it takes to complete a backup or restore operation by choosing an optimal value for the number of buffers, the buffer size and the parallelism settings. The values are based on the amount of utility heap memory available, the number of processors available and the database configuration.
- To reduce the amount of time required to complete a restore operation, use multiple source devices.
- If a table contains large amounts of long field and LOB data, restoring it could be very time consuming. If the database is enabled for rollforward recovery, the **RESTORE** command provides the capability to restore selected table spaces. If the long field and LOB data is critical to your business, restoring these table spaces should be considered against the time required to complete the backup task for these table spaces. By storing long field and LOB data in separate table spaces, the time required to complete the restore operation can be reduced by choosing not to restore the table spaces containing the long field and LOB data. If the LOB data can be reproduced from a separate source, choose the NOT LOGGED option when creating or altering a table to include LOB columns. If you choose not to restore the table spaces that contain long field and LOB data, but you need to restore the table spaces that contain the table, you must roll forward to the end of the logs so that all table spaces that contain table data are consistent.

  Note: If you back up a table space that contains table data without the associated long or LOB fields, you cannot perform point-in-time rollforward recovery on that table space. All the table spaces for a table must be rolled forward simultaneously to the same point in time.
- The following apply for both backup and restore operations:
  - Multiple devices should be used.
  - Do not overload the I/O device controller bandwidth.
- DB2 database products use multiple agents to perform both crash recovery and database rollforward recovery. You can expect better performance during these operations, particularly on symmetric multi-processor (SMP) machines; using multiple agents during database recovery takes advantage of the extra CPUs that are available on SMP machines.

  The agent type introduced by parallel recovery is **db2agnsc**. DB2 database managers choose the number of agents to be used for database recovery based on the number of CPUs on the machine.

  DB2 database managers distribute log records to these agents so that they can be reapplied concurrently, where appropriate. For example, the processing of log records associated with insert, delete, update, add key, and delete key operations can be parallelized in this way. Because the log records are parallelized at the page level (log records on the same data page are processed by the same agent), performance is enhanced, even if all the work was done on one table.
- When you perform a recover operation, DB2 database managers will automatically choose an optimal value for the number of buffers, the buffer size

and the parallelism settings. The values will be based on the amount of utility heap memory available, the number of processors available and the database configuration. Therefore, depending on the amount of storage available on your system, you should consider allocating more memory by increasing the **util_heap_sz** configuration parameter.

Chapter 47. RESTORE DATABASE command

Use the RESTORE DATABASE command to restore the DB2 database to a previous state. A backup image of the database must exist before you can use this command.

The simplest form of the DB2 **RESTORE DATABASE** command requires only that you specify the alias name of the database that you want to restore. For example:

```
db2 restore db sample
```

In this example, because the SAMPLE database exists and will be replaced when the **RESTORE DATABASE** command is issued, the following message is returned:

```
SQL2539W  Warning!  Restoring to an existing database that is the same as
the backup image database.  The database files will be deleted.
Do you want to continue ? (y/n)
```

If you specify `y`, the restore operation should complete successfully.

A database restore operation requires an exclusive connection: that is, no applications can be running against the database when the operation starts, and the restore utility prevents other applications from accessing the database until the restore operation completes successfully. A table space restore operation, however, can be done online.

A table space is not usable until the restore operation (possibly followed by rollforward recovery) completes successfully.

If you have tables that span more than one table space, you should back up and restore the set of table spaces together.

When doing a partial or subset restore operation, you can use either a table space-level backup image, or a full database-level backup image and choose one or more table spaces from that image. All the log files associated with these table spaces from the time that the backup image was created must exist.

You can restore a database from a backup image taken on a 32-bit level into a 64-bit level, but not vice versa.

If you are restoring backups from 32-bit level environments to 64-bit level environments, review your database configuration parameters to ensure that they are optimized for the 64-bit instance environment. For example, the statement heap's default value is lower in 32-bit environments than in 64-bit environments.

The DB2 backup and restore utilities should be used to backup and restore your databases. Moving a fileset from one machine to another is not recommended as this may compromise the integrity of the database.

Under certain conditions, you can use transportable sets with the **RESTORE DATABASE** command to move databases. .

In IBM Data Studio Version 3.1 or later, you can use the task assistant for restoring database backups. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Privileges, authorities, and authorization required to use restore

You must have SYSADM, SYSCTRL, or SYSMAINT authority to restore to an *existing* database from a full database backup. To restore to a *new* database, you must have SYSADM or SYSCTRL authority.

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects.

Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

Implications for restoring databases

The **RESTORE DATABASE** command is used to restore a database from a backup image.

During a restore operation it is possible to choose the location of the database path, and it is also possible to redefine the storage paths that are associated with the storage groups. The database path and the storage paths are set by using a combination of **TO**, **ON**, and **DBPATH ON** with the **RESTORE DATABASE** command, or using the **SET STOGROUP PATHS** command.

For example, here are some valid **RESTORE** commands:

```
RESTORE DATABASE TEST1
RESTORE DATABASE TEST2 TO X:
RESTORE DATABASE TEST3 DBPATH ON D:
RESTORE DATABASE TEST3 ON /path1, /path2, /path3
RESTORE DATABASE TEST4 ON E:\newpath1, F:\newpath2 DBPATH ON D:
```

As it does in the case of the **CREATE DATABASE** command, the database manager extracts the following two pieces of information that pertain to storage locations:

- The *database path* (which is where the database manager stores various control files for the database)
  - If **TO** or **DBPATH ON** is specified, this indicates the database path.
  - If **ON** is used but **DBPATH ON** is not specified with it, the first path listed with **ON** is used as the database path (in addition to it being a storage path).
  - If none of **TO**, **ON**, or **DBPATH ON** is specified, the **dftdbpath** database manager configuration parameter determines the database path.

  Note: If a database with the same name exists on disk, the database path is ignored, and the database is placed into the same location as the existing database.
- The *storage paths* of each *storage group* (where the database manager creates automatic storage table space containers)
  - If **ON** is specified, all of the paths listed are considered storage paths, and these paths are used instead of the ones stored within the backup image. If the database contains multiple storage groups, every defined storage group uses the new storage group paths.

- If the **SET STOGROUP PATHS** command is used, the storage paths provided are used for the specified storage group instead of the ones stored within the backup image.
- If **ON** is not specified and the **SET STOGROUP PATHS** command is not used, no change is made to the storage paths (the storage paths stored within the backup image are maintained).

To make this concept clearer, the same five **RESTORE** command examples presented previously are shown in the following table with their corresponding storage paths:

Table 125. Restore implications regarding database and storage paths

	No database with the same name exists on disk		Database with the same name exists on disk	
RESTORE DATABASE command	Database path	Storage paths	Database path	Storage paths
`RESTORE DATABASE TEST1`	**dftdbpath**	Uses storage paths of the existing database	Uses database path of existing database	Uses storage paths defined in the backup image
`RESTORE DATABASE TEST2 TO X:`	`X:`	Uses storage paths of the existing database	Uses database path of existing database	Uses storage paths defined in the backup image

`RESTORE DATABASE TEST3`
`DBPATH ON /db2/databases`
	`/db2/databases`	Uses storage paths of the existing database	Uses database path of existing database	Uses storage paths defined in the backup image

`RESTORE DATABASE TEST4`
`ON /path1, /path2, /path3`
	`/path1`	All storage groups use `/path1`, `/path2`, `/path3` for their storage paths	Uses database path of existing database	All storage groups use `/path1`, `/path2`, `/path3`for their storage paths

`RESTORE DATABASE TEST5`
`ON E:\newpath1, F:\newpath2`
`DBPATH ON D:`
	`D:`	All storage groups use `E:\newpath1`, `F:\newpath2` for their storage paths	Uses database path of existing database	All storage groups use `E:\newpath1`, `F:\newpath2` for their storage paths

For those cases where storage paths have been redefined as part of the restore operation, the table spaces that are defined to use automatic storage are automatically redirected to the new paths. However, you cannot explicitly redirect containers associated with automatic storage table spaces using the **SET TABLESPACE CONTAINERS** command; this action is not permitted.

Use the **-s** option of the **db2ckbkp** command to show whether storage groups exist for a database within a backup image. The storage groups and their storage paths are displayed.

For multi-partition databases, the **RESTORE DATABASE** command has a few extra implications:

1. The database must use the same set of storage paths on all database partitions.
2. Issuing a **RESTORE** command with new storage paths can be done only on the catalog database partition, which sets the state of the database to RESTORE_PENDING on all non-catalog database partitions.

Table 126. Restore implications for multi-partition databases

RESTORE DATABASE command	Issued on database partition #	No database with the same name exists on disk		Database with the same name exists on disk (includes skeleton databases)	
		Result on other database partitions	Storage paths	Result on other database partitions	Storage paths
RESTORE DATABASE TEST1	Catalog database partition	A skeleton database is created using the storage paths from the backup image on the catalog database partition. All other database partitions are placed in a RESTORE_ PENDING state.	Uses storage paths defined in the backup image	Nothing. Storage paths have not changed so nothing happens to other database partitions	Uses storage paths defined in the backup image
	Non-catalog database partition	SQL2542N or SQL2551N is returned. If no database exists, the catalog database partition must be restored first.	N/A	Nothing. Storage paths have not changed so nothing happens to other database partitions	Uses storage paths defined in the backup image
RESTORE DATABASE TEST2 ON /path1, /path2, /path3	Catalog database partition	A skeleton database is created using the storage paths specified in the **RESTORE** command. All other database partitions are place in a RESTORE_ PENDING state.	All storage groups use /path1, /path2, /path3 for their storage paths		All storage groups use /path1, /path2, /path3 for their storage paths
	Non-catalog database partition	SQL1174N is returned. If no database exists, the catalog database partition must be restored first. Storage paths cannot be specified on the **RESTORE** of a non-catalog database partition.	N/A	SQL1172N is returned. New storage paths cannot be specified on the **RESTORE** of a non-catalog database partition.	N/A

Using restore

Use the **RESTORE DATABASE** command to recover a database or table space after a problem such as media or storage failure, or application failure. If you have backed up your database, or individual table spaces, you can recreate them if they have become damaged or corrupted in some way.

Before you begin

When restoring to an *existing* database, you should not be connected to the database that is to be restored: the restore utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the restore operation. When restoring to a *new* database, an instance attachment is required to create the database. When restoring to a *new remote* database, you must first attach to the instance where the new database will reside. Then, create the new database, specifying the code page and the territory of the server. Restore will overwrite the code page of the destination database with the codepage of the backup image.

About this task

The database can be local or remote.

The following restrictions apply to the restore utility:

- You can only use the restore utility if the database has been previously backed up using the DB2 backup utility.
- If users other than the instance owner (on UNIX), or members of the DB2ADMNS or Administrators group (on Windows) attempt to restore a backup image, they will get an error (SQL2061N). If other users need access to the backup image, the file permissions need to be changed after the backup is generated.
- A database restore operation cannot be started while the rollforward process is running.
- If you do not specify the **TRANSPORT** option, then you can restore a table space into an existing database only if the table space currently exists, and if it is the same table space. In this situation, “same” means that the table space was not dropped and then recreated between the backup and the restore operation. The database on disk and in the backup image must be the same.
- You cannot issue a table space-level restore of a table space-level backup to a new database.
- You cannot perform an online table space-level restore operation involving the system catalog tables.
- You cannot restore a backup taken in a single database partition environment into an existing partitioned database environment. Instead you must restore the backup to a single database partition environment and then add database partitions as required.
- When restoring a backup image with one code page into a system with a different codepage, the system code page will be overwritten by the code page of the backup image.
- You cannot use the **RESTORE DATABASE** command to convert nonautomatic storage enabled table spaces to automatic storage enabled table space.
- The following restrictions apply when the **TRANSPORT** option is specified:
  - If the backup image can be restored by a restore operation, and is supported for upgrades, then it can be transported.
  - If an online backup is used, then both source and target data servers must be running the same DB2 version.
  - The **RESTORE DATABASE** command must be issued against the target database. If the remote client is of the same platform as the server, then schema transport can be executed locally on the server or through remote instance attachment. If a target database is a remote database cataloged in the instance where transport runs locally, then schema transport against that remote target database is not supported.
  - You can only transport tables spaces and schemas into an existing database. The transport operation will not create a new database. To restore a database into a new database, you can use the **RESTORE DATABASE** command without specifying the **TRANSPORT** option.

  - If the schemas in the source database are protected by any DB2 security settings or authorizations, then the transported schemas in the target database will retain these same settings or authorizations.
  - Transport is not supported for partitioned database environments.
  - If any of the tables within the schema contains an XML column, the transport fails.
  - The **TRANSPORT** option is incompatible with the REBUILD option.
  - The **TRANSPORT** option is not supported for restore from a snapshot backup image.
  - The target database must be enabled for database recovery.
  - The staging database is created for transport. It cannot be used for other operations.
  - The database configuration parameters on the staging table and the target table need to be the same, or the transport operation fails with an incompatibility error.
  - The **auto_reval** configuration parameter must be set to deferred_force on the target database to transport objects listed as invalid. Otherwise, the transport fails.
  - If an online backup image is used, and the active logs are not included, then the transport operation fails.
  - If an online backup is used, then the backup image must have been created with the INCLUDE LOGS option
  - If the backup image is from a previous release, it must be a full offline database level backup image.
  - If an error occurs on either the staging or target database, the entire restore operation must be reissued. All failures that occur are logged in the **db2diag** log file on the target server and should be reviewed before reissuing the **RESTORE** command.
  - If the transport client fails, then the staging database might not be properly cleaned up. In this case, you need to drop the staging database. Before re-issuing the **RESTORE** command, drop all staging databases to prevent containers of staging database from blocking subsequent transport.
  - Concurrent transport running against the same target database is not supported.
  - Generating a redirected restore script is not supported with table space transport.
- You can restore a table space if the storage group has been updated. The target storage group during the table space restore is the storage group the table space is currently associated with when **RESTORE** is executed.
- You cannot perform a point-in-time recovery to an earlier storage group association.

Procedure

To invoke the restore utility:

- Issue the **RESTORE DATABASE** command.
- Call the db2Restore application programming interface (API).
- Open the task assistant in IBM Data Studio for the **RESTORE DATABASE** command.

Example

Following is an example of the **RESTORE DATABASE** command issued through the CLP:

```
db2 restore db sample from D:\DB2Backups taken at 20010320122644
```

Restoring from a snapshot backup image

Restoring from a snapshot backup uses the fast copying technology of a storage device to perform the data copying portion of the restore.

Before you begin

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, refer to this tech note.

You must perform a snapshot backup before you can restore from a snapshot backup. See: "Performing a snapshot backup" on page 759.

Procedure

You can restore from a snapshot backup using the **RESTORE DATABASE** command with the **USE SNAPSHOT** parameter, or the db2Restore API with the SQLU_SNAPSHOT_MEDIA media type:

- **RESTORE DATABASE** command:

  ```
  db2 restore db sample use snapshot
  ```

- db2Restore API:

  ```
  int sampleRestoreFunction( char dbAlias[],
                             char restoredDbAlias[],
                             char user[],
                             char pswd[],
                             char workingPath[] )
  {
    db2MediaListStruct mediaListStruct = { 0 };

    rmediaListStruct.locations = &workingPath;
    rmediaListStruct.numLocations = 1;
    rmediaListStruct.locationType = SQLU_SNAPSHOT_MEDIA;


    db2RestoreStruct restoreStruct = { 0 };

    restoreStruct.piSourceDBAlias = dbAlias;
    restoreStruct.piTargetDBAlias = restoredDbAlias;
    restoreStruct.piMediaList = &mediaListStruct;
    restoreStruct.piUsername = user;
    restoreStruct.piPassword = pswd;
    restoreStruct.iCallerAction = DB2RESTORE_STORDEF_NOINTERRUPT;


    struct sqlca sqlca = { 0 };


    db2Restore(db2Version900, &restoreStruct, &sqlca);

    return 0;
  }
  ```

Restoring to an existing database

For a database-level restore, the backup image can differ from the existing database in its alias name, its database name, or its database *seed*. A database seed is a unique identifier for a database that does not change during the life of the database.

The database manager assigns the seed when you create the database. DB2 always uses the seed from the backup image.You can restore a table space into an existing database only if the table space exists and if the table spaces are the same, meaning that you did not drop the table space and then re-create it between the backup and the restore operations. The database on disk and in the backup image must be the same.You cannot modify the currently defined storage groups or explicitly create new storage groups when restoring a table space.

When restoring to an existing database, the restore utility performs the following actions:

- Deletes table, index, and long field data from the existing database and replaces it with data from the backup image.
- Replaces table entries for each table space that you are restoring.
- Retains the recovery history file unless it is damaged or has no entries. If the recovery history file is damaged or contains no entries, the database manager copies the file from the backup image. If you want to replace the recovery history file, you can issue the **RESTORE DATABASE** command with the **REPLACE HISTORY FILE** parameter.
- Retains the authentication type for the existing database.
- Retains the database directories for the existing database. The directories define where the database is located and how it is cataloged.
- Compares the database seeds. If the seeds are different, the utility performs the following actions:
  - Deletes the logs that are associated with the existing database.
  - Copies the database configuration file from the backup image.
  - If you specify the **NEWLOGPATH** parameter, the utility sets the **NEWLOGPATH** parameter for the **RESTORE DATABASE** command to the value of the **logpath** database configuration parameter. If you do not specify the **NEWLOGPATH** parameter, the utility performs the following actions:
    - Validates the log path.
    - If the database cannot use the log path, the utility changes the database configuration to use the default log path.

  If the database seeds are the same, the utility performs the following actions:
  - Deletes all log files if the image is for a non-recoverable database.
  - Deletes empty log files if the image is for a recoverable database. Non-empty log files are not affected.
  - Retains the current database configuration file.
  - If you specify the **NEWLOGPATH** parameter, the utility sets the **NEWLOGPATH** parameter for the **RESTORE DATABASE** command to the value of the **logpath** database configuration parameter. If you do not specify the **NEWLOGPATH** parameter, the utility performs the following actions:
    - Copies the current log path to the database configuration file.
    - Validates the log path.

- If the database cannot use the log path, the utility changes the database configuration to use the default log path.

Restoring to a new database

You can create a new database and then restore a full database backup image to it. If you do not create a new database, the restore utility creates one.

When restoring to a new database, the restore utility:

- Creates a new database, using the database alias name that was specified through the target database alias parameter. (If a target database alias was not specified, the restore utility creates the database with an alias that is the same as that specified through the source database alias parameter.)
- Restores the database configuration file from the backup image.
- Sets **NEWLOGPATH** to the value of the **logpath** database configuration parameter if **NEWLOGPATH** was specified on the **RESTORE DATABASE** command. Validates the log path: If the path cannot be used by the database, changes the database configuration to use the default log path.
- Restores the authentication type from the backup image.
- Restores the comments from the database directories in the backup image.
- Restores the recovery history file for the database.
- Overwrites the code page of the database with the codepage of the backup image.

Using incremental restore in a test and production environment

Once a production database is enabled for incremental backup and recovery, you can use an incremental or delta backup image to create or refresh a test database.

You can do this by using either manual or automatic incremental restore.

To restore the backup image from the production database to the test database, use the INTO *target-database-alias* option on the **RESTORE DATABASE** command. For example, in a production database with the following backup images:

```
backup db prod
Backup successful. The timestamp for this backup image is : ts1

backup db prod incremental
Backup successful. The timestamp for this backup image is : ts2
```

an example of a manual incremental restore would be:

```
restore db prod incremental taken at ts2 into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.

restore db prod incremental taken at ts1 into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.

restore db prod incremental taken at ts2 into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.
```

If the database TEST already exists, the restore operation overwrites any data that is already there. If the database TEST does not exist, the restore utility creates it and then populates it with the data from the backup images.

Since automatic incremental restore operations are dependent on the database history, the restore steps change slightly based on whether the test database exists. To perform an automatic incremental restore to the database TEST, its history must contain the backup image history for database PROD. The database history for the backup image replaces any database history that already exists for database TEST if either of the following are true:

- The database TEST does not exist when the **RESTORE DATABASE** command is issued.
- The database TEST exists when the **RESTORE DATABASE** command is issued, and the database TEST history contains no records.

The following example shows an automatic incremental restore to database TEST which does not exist:

```
restore db prod incremental automatic taken at ts2 into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.
```

The restore utility creates the TEST database and populates it.

If the database TEST does exist and the database history is not empty, you must drop the database before the automatic incremental restore operation as follows:

```
drop db test
DB20000I  The DROP DATABASE command completed successfully.

restore db prod incremental automatic taken at ts2 into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.
```

If you do not want to drop the database, you can issue the **PRUNE HISTORY** command with a timestamp far into the future and the **WITH FORCE OPTION** parameter before issuing the **RESTORE DATABASE** command:

```
connect to test
Database Connection Information

Database server           = server_id
SQL authorization ID      = id
Local database alias      = TEST

prune history 9999 with force option
DB20000I  The PRUNE command completed successfully.

connect reset
DB20000I  The SQL command completed successfully.
restore db prod incremental automatic taken at ts2 into test without
prompting
SQL2540W  Restore is successful, however a warning "2539" was
encountered during Database Restore while processing in No
Interrupt mode.
```

In this case, the **RESTORE DATABASE** command acts in the same manner as when the database TEST did not exist.

If the database TEST does exist and the database history is empty, you do not have to drop the database TEST before the automatic incremental restore operation:

```
restore db prod incremental automatic taken at ts2 into test without
prompting
SQL2540W  Restore is successful, however a warning "2539" was
encountered during Database Restore while processing in No
Interrupt mode.
```

You can continue taking incremental or delta backups of the test database without first taking a full database backup. However, if you ever need to restore one of the incremental or delta images you have to perform a manual incremental restore. This requirement is because automatic incremental restore operations require that each of the backup images restored during an automatic incremental restore are created from the same database alias.

If you make a full database backup of the test database after you complete the restore operation using the production backup image, you can take incremental or delta backups and can restore them using either manual or automatic mode.

Performing a redirected restore operation

A database restore operation uses a database backup image to recreate a database.

Use a redirected restore operation in any of the following situations:

- If you want to restore a backup image to a target machine that is different from the source machine
- If you want to restore your table space containers into a different physical location
- If your restore operation failed because one or more containers are inaccessible
- If you want to redefine the paths of a defined storage group

Restrictions:

You cannot use a redirected restore to move data from one operating system to another.

You cannot create or drop a storage group during the restore process.

You cannot modify storage group paths during a table space restore process even if you are restoring all table spaces that are associated with the storage group.

The process for performing a redirected restore by using an incremental backup image is similar to the process of performing a redirected restore by using a non-incremental backup image. Use one of the following approaches:

- Issue the **RESTORE DATABASE** command with the **REDIRECT** parameter, and specify the backup image to use for the incremental restore of the database.
- Generate a redirected restore script from a backup image, and then modify the script as required.

Using the **RESTORE DATABASE** command approach is a two-step database restore process with an intervening step for defining a table space container or storage group path. To perform a redirected restore:

1. Issue the **RESTORE DATABASE** command with the **REDIRECT** parameter.
2. Take one of the following steps:
   - Define table space containers by issuing the **SET TABLESPACE CONTAINERS** command.
   - Define storage group paths for the database to be restored by issuing the **SET STOGROUP PATHS** command.
3. Issue the **RESTORE DATABASE** command again, this time specifying the **CONTINUE** parameter.

After you issue the **RESTORE CONTINUE** command, the new path takes effect as the table space container path for all associated table spaces. If you issue a **LIST**

TABLESPACE CONTAINERS command or a **GET SNAPSHOT FOR TABLESPACES** command after the **SET STOGROUP PATHS** command and before the **RESTORE CONTINUE** command, the output for the table space container paths does not reflect the new paths that you specified by using the **SET STOGROUP PATHS** command.

During a redirected restore operation, directory and file containers are automatically created if they do not exist. The database manager does not automatically create device containers.

DB2 database products provide SQL statements for adding, changing, or removing table space containers non-automatic-storage DMS table spaces, and storage group paths of automatic storage table spaces. A redirected restore is the only way to modify a non-automatic-storage SMS table space container configuration.

You can redefine table space containers or modify storage group paths by issuing the **RESTORE DATABASE** command with the **REDIRECT** parameter.

Table space container redirection provides considerable flexibility for managing table space containers. You can alter the storage group configuration of a database before restoring any data pages from the backup image, similar to the way that you can redirect table space container paths. If you renamed a storage group since you produced the backup image, the storage group name that is specified by the **SET STOGROUP PATHS** command refers to the storage group name from the backup image, not the more recent name.

Performing a redirected restore operation in a partitioned database environment

In a partitioned database environment, during a redirected database restore, you can redirect the storage group paths to new storage group paths only from the catalog database partition. For all other database partitions you must have their storage group paths synchronized with those of the catalog partition.

Modifying any storage group paths on the catalog partition places all non-catalog partitions into a RESTORE_PENDING state. If you redirect storage group paths, you must restore the catalog partition before any other database partition. After you restore the catalog database partition, you can restore the non-catalog database partitions in parallel, without any storage group path redirection. The non-catalog database partitions automatically acquire the new storage group paths that you specified for the catalog database partition. New storage group paths are also automatically acquired when the storage group paths are implicitly changed during a database restore when you are restoring a different database (one with a different name, instance, or seed).

If you modified the storage group paths since taking the last backup, you can still use that backup image (with different storage group paths) for a restore on any database partition. This restore is not considered a redirected restore. Restoring from that backup image temporarily causes the database partition to use the storage group paths that you defined at the time that you created the backup. Perform a rollforward recovery to reapply the storage group path modifications and resynchronize all of the database partitions.

Examples

Example 1

You can perform a table space container redirected restore on database SAMPLE by using the **SET TABLESPACE CONTAINERS** command to define table space containers:

```
db2 restore db sample redirect without prompting
SQL1277W A redirected restore operation is being performed.
During a table space restore, only table spaces being restored can
have their paths reconfigured. During a database restore, storage
group storage paths and DMS table space containers can be reconfigured.

DB20000I The RESTORE DATABASE command completed successfully.

db2 set tablespace containers for 2 using (path 'userspace1.0', path
'userspace1.1')
DB20000I The SET TABLESPACE CONTAINERS command completed successfully.

db2 restore db sample continue
DB20000I The RESTORE DATABASE command completed successfully.
```

Example 2

You can redefine the paths of the defined storage group by using the **SET STOGROUP PATHS** command:

```
RESTORE DB SAMPLE REDIRECT

SET STOGROUP PATHS FOR sg_hot ON '/ssd/fs1', '/ssd/fs2'
SET STOGROUP PATHS FOR sg_cold ON '/hdd/path1', '/hdd/path2'

RESTORE DB SAMPLE CONTINUE
```

Example 3

Following is a typical non-incremental redirected restore scenario for a database whose alias is MYDB:

1. Issue a RESTORE DATABASE command with the REDIRECT option.
   ```
   db2 restore db mydb replace existing redirect
   ```
2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers you want to redefine. For example, in a Windows environment:
   ```
   db2 set tablespace containers for 5 using
          (file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
   ```

   To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command for every table space whose container locations are being redefined.
3. After successful completion of steps 1 and 2, issue:
   ```
   db2 restore db mydb continue
   ```
   This is the final step of the redirected restore operation.
4. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

Note:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:
   ```
   db2 restore db mydb abort
   ```
2. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

Example 4

Following is a typical manual incremental redirected restore scenario for a database whose alias is MYDB and has the following backup images:

```
backup db mydb
Backup successful. The timestamp for this backup image is : <ts1>

backup db mydb incremental
Backup successful. The timestamp for this backup image is : <ts2>
```

1. Issue a RESTORE DATABASE command with the INCREMENTAL and REDIRECT options.

   ```
   db2 restore db mydb incremental taken at <ts2> replace existing redirect
   ```

2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example, in a Windows environment:

   ```
   db2 set tablespace containers for 5 using
          (file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
   ```

   To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.

3. After successful completion of steps 1 and 2, issue:

   ```
   db2 restore db mydb continue
   ```

4. The remaining incremental restore commands can now be issued as follows:

   ```
   db2 restore db mydb incremental taken at <ts1>
   db2 restore db mydb incremental taken at <ts2>
   ```

   This is the final step of the redirected restore operation.

Note:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

   ```
   db2 restore db mydb abort
   ```

2. After successful completion of step 3, and before issuing all the required commands in step 4, the restore operation can be aborted by issuing:

   ```
   db2 restore db mydb incremental abort
   ```

3. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.
4. If either restore command fails in step 4, the failing command can be reissued to continue the restore process.

Example 5

Following is a typical automatic incremental redirected restore scenario for the same database:

1. Issue a RESTORE DATABASE command with the INCREMENTAL AUTOMATIC and REDIRECT options.

   ```
   db2 restore db mydb incremental automatic taken at <ts2>
          replace existing redirect
   ```

2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example, in a Windows environment:

   ```
   db2 set tablespace containers for 5 using
          (file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
   ```

To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.

3. After successful completion of steps 1 and 2, issue:

   ```
   db2 restore db mydb continue
   ```

   This is the final step of the redirected restore operation.

Note:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

   ```
   db2 restore db mydb abort
   ```
2. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1 after issuing:

   ```
   db2 restore db mydb incremental abort
   ```

Redefine table space containers by restoring a database using an automatically generated script

When you restore a database, the restore utility assumes that the physical container layout will be identical to that of the database when it was backed up.

If you need to change the location or size of any of the physical containers, you must issue the **RESTORE DATABASE** command with the **REDIRECT** option. Using this option requires that you specify the locations of physical containers stored in the backup image and provide the complete set of containers for each non-automatic table space that will be altered. You can capture the container information at the time of the backup, but this can be cumbersome.

To make it easier to perform a redirected restore, the restore utility allows you to generate a redirected restore script from an existing backup image by issuing the **RESTORE DATABASE** command with the **REDIRECT** parameter and the **GENERATE SCRIPT** parameter. The restore utility examines the backup image, extracts container information from the backup image, and generates a CLP script that includes all of the detailed container information. You can then modify any of the paths or container sizes in the script, then run the CLP script to recreate the database with the new set of containers. The script you generate can be used to restore a database even if you only have a backup image and you do not know the layout of the containers. The script is created on the client. Using the script as your basis, you can decide where the restored database will require space for log files and containers and you can change the log file and container paths accordingly.

The generated script consists of four sections:

Initialization

The first section sets command options and specifies the database partitions on which the command will run. The following is an example of the first section:

```
UPDATE COMMAND OPTIONS USING S ON Z ON SAMPLE_NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM 0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
```

where

- S ON specifies that execution of the command should stop if a command error occurs

- Z ON SAMPLE_NODE0000.out specifies that output should be directed to a file named *dbalias*_NODE*dbpartitionnum*.out
- V ON specifies that the current command should be printed to standard output.

  When running the script on a partitioned database environment, it is important to specify the database partition on which the script commands will run.

RESTORE DATABASE command with the REDIRECT parameter

The second section starts the **RESTORE DATABASE** command and uses the **REDIRECT** parameter. This section can use all of the **RESTORE DATABASE** command parameters, except any parameters that cannot be used with the **REDIRECT** parameter. The following is an example of the second section:

```
RESTORE DATABASE SAMPLE
-- USER 'username'
-- USING 'password'
FROM '/home/jseifert/backups'
TAKEN AT 20050906194027
-- DBPATH ON 'target-directory'
INTO SAMPLE
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00001/LOGSTREAM0000/'
-- WITH num-buff BUFFERS
-- BUFFER buffer-size
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM n
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
```

Table space definitions

This section contains table space definitions for each table space in the backup image or specified on the command line. There is a section for each table space, consisting of a comment block that contains information about the name, type and size of the table space. The information is provided in the same format as a table space snapshot. You can use the information provided to determine the required size for the table space. In cases where you are viewing output of a table space created using automatic storage, you will not see a SET TABLESPACE CONTAINERS clause. The following is an example of the table space definition section:

```
-- *********************************************************************
-- ** Tablespace name                            = SYSCATSPACE
-- **   Tablespace ID                            = 0
-- **   Tablespace Type                          = System managed space
-- **   Tablespace Content Type                  = Any data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Total number of pages                    = 5572
-- *********************************************************************
SET TABLESPACE CONTAINERS FOR 0
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0000.0'
);
-- *********************************************************************
-- ** Tablespace name                            = TEMPSPACE1
-- **   Tablespace ID                            = 1
-- **   Tablespace Type                          = System managed space
-- **   Tablespace Content Type                  = System Temporary data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Total number of pages                    = 0
-- *********************************************************************
SET TABLESPACE CONTAINERS FOR 1
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0001.0'
);
-- *********************************************************************
-- ** Tablespace name                            = DMS
-- **   Tablespace ID                            = 2
-- **   Tablespace Type                          = Database managed space
-- **   Tablespace Content Type                  = Any data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Auto-resize enabled                      = No
-- **   Total number of pages                    = 2000
-- **   Number of usable pages                   = 1960
-- **   High water mark (pages)                  = 96
-- *********************************************************************
SET TABLESPACE CONTAINERS FOR 2
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   '/tmp/dms1'                                           1000
, FILE   '/tmp/dms2'                                           1000
);
```

RESTORE DATABASE command with the CONTINUE parameter

The final section issues the **RESTORE DATABASE** command with the **CONTINUE** parameter, to complete the redirected restore. The following is an example of the final section:

```
RESTORE DATABASE SAMPLE CONTINUE;
```

Performing a redirected restore using an automatically generated script

When you perform a redirected restore operation, you must specify the locations of physical containers that are stored in the backup image and provide the complete set of containers for each table space that you are altering.

Before you begin

You can perform a redirected restore only if the database was previously backed up using the DB2 backup utility.

About this task

- If the database exists, you must be able to connect to it in order to generate the script. Therefore, if the database requires an upgrade or crash recovery, this must be done before you attempt to generate a redirected restore script.
- If you are working in a partitioned database environment, and the target database does not exist, you cannot run the command to generate the redirected restore script concurrently on all database partitions. Instead, the command to generate the redirected restore script must be run one database partition at a time, starting from the catalog partition.

  Alternatively, you can first create a dummy database with the same name as your target database. After the dummy database is created, you can then generate the redirected restore script concurrently on all database partitions.

- Even if you specify the **REPLACE EXISTING** parameter when you issue the **RESTORE DATABASE** command to generate the script, the **REPLACE EXISTING** parameter is commented out in the script.
- For security reasons, your password does not appear in the generated script. You need to enter the password manually.
- The restore script includes the storage group associations for every table space that you restore.

Procedure

To perform a redirected restore using a script:

1. Use the restore utility to generate a redirected restore script. The restore utility can be invoked through the command line processor (CLP) or the db2Restore application programming interface (API). The following is an example of the **RESTORE DATABASE** command with the **REDIRECT** parameter and the **GENERATE SCRIPT** parameter:
   ```
   db2 restore db test from /home/jseifert/backups taken at 20050304090733
         redirect generate script test_node0000.clp
   ```
   This creates a redirected restore script on the client called `test_node0000.clp`.
2. Open the redirected restore script in a text editor to make any modifications that are required. You can modify:
   - Restore options
   - Automatic storage paths
   - Container layout and paths
3. Run the modified redirected restore script. For example:
   ```
   db2 -tvf test_node0000.clp
   ```

Cloning a production database using different storage group paths

You might have to clone a production database onto a test database that uses a different machine. The test machine and production server are likely to have different storage group paths. The test system might not have paths backed by the newest and fastest storage disks.

About this task

Suppose you have a production database `proddb`, where some data is in storage group `sg_hot`, which has paths on an SSD device. You want to restore the data into the less expensive and lower-performance test database `testdb`. The test system does not have the SSD device, but the other paths are equivalent. Performing a redirected restore can change the paths for `sg_hot` on the test system without changing the other storage groups.

Procedure

To restore data from a production database to a test database:

1. Back up the production database. Issue the following command:
   ```
   BACKUP DATABASE production_db TO /backup
   ```
   where *production_db* is the production database.
2. Set up a redirected restore to the test database. Issue the following command:

```
RESTORE DATABASE testdb REDIRECT
```

where *testdb* is the test database.

3. Modify the storage paths for `sg_hot` because no hot storage is available on the test database. Issue the following command:
   ```
   SET STOGROUP PATHS FOR sg_hot ON '/hdd/path1', '/hdd/path2'
   ```
   where *sg_hot* is the `sg_hot` storage group.
4. Proceed with the test database restore. Issue the following command:
   ```
   RESTORE DATABASE testdb CONTINUE
   ```
5. Update the storage group name to correspond with the new paths. Use the following commands:
   ```
   CONNECT TO testdb
   RENAME STOGROUP sg_hot TO sg_cold
   ```

Database rebuild

Rebuilding a database is the process of restoring a database or a subset of its table spaces using a set of restore operations. The functionality provided with database rebuild makes DB2 database products more robust and versatile, and provides you with a more complete recovery solution.

The ability to rebuild a database from table space backup images means that you no longer have to take as many full database backups. As databases grow in size, opportunities for taking a full database backup are becoming limited. With table space backup as an alternative, you no longer need to take full database backups as frequently. Instead, you can take more frequent table space backups and plan to use them, along with log files, in case of a disaster.

In a recovery situation, if you need to bring a subset of table spaces online faster than others, you can use rebuild to accomplish this. The ability to bring only a subset of table spaces online is especially useful in a test and production environment.

Rebuilding a database involves a series of potentially many restore operations. A rebuild operation can use a database image, or table space images, or both. It can use full backups, or incremental backups, or both. The initial restore operation restores the target image, which defines the structure of the database that can be restored (such as the table space set, the storage groups and the database configuration). For recoverable databases, rebuilding allows you to build a database that is connectable and that contains the subset of table spaces that you need to have online, while keeping table spaces that can be recovered at a later time offline.

The method you use to rebuild your database depends on whether it is recoverable or non-recoverable.

- If the database is recoverable, use one of the following methods:
  - Using a full or incremental database or table space backup image as your target, rebuild your database by restoring SYSCATSPACE and any other table spaces from the target image only using the **REBUILD** option. You can then roll your database forward to a point in time.
  - Using a full or incremental database or table space backup image as your target, rebuild your database by specifying the set of table spaces defined in the database at the time of the target image to be restored using the **REBUILD**

option. SYSCATSPACE must be part of this set. This operation will restore those table spaces specified that are defined in the target image and then use the recovery history file to find and restore any other required backup images for the remaining table spaces not in the target image automatically. Once the restores are complete, roll your database forward to a point in time.

- If the database is non-recoverable:
  - Using a full or incremental database backup image as your target, rebuild your database by restoring SYSCATSPACE and any other table spaces from the target image using the appropriate **REBUILD** syntax. When the restore completes you can connect to the database.

Specifying the target image

To perform a rebuild of a database, you start by issuing the **RESTORE** command, specifying the most recent backup image that you use as the target of the restore operation. This image is known as the target image of the rebuild operation, because it defines the structure of the database to be restored, including the table spaces that can be restored, the database configuration, and the log sequence. The rebuild target image is specified using the **TAKEN AT** parameter in the **RESTORE DATABASE** command. The target image can be any type of backup (full, table space, incremental, online or offline). The table spaces defined in the database at the time the target image was created will be the table spaces available to rebuild the database.

You must specify the table spaces you want restored using one of the following methods:

- Specify that you want all table spaces defined in the database to be restored and provide an exception list if there are table spaces you want to exclude
- Specify that you want all table spaces that have user data in the target image to be restored and provide an exception list if there are table spaces you want to exclude
- Specify the list of table spaces defined in the database that you want to restore

Once you know the table spaces you want the rebuilt database to contain, issue the **RESTORE** command with the appropriate **REBUILD** option and specify the target image to be used.

Rebuild phase

After you issue the **RESTORE** command with the appropriate **REBUILD** option and the target image has been successfully restored, the database is considered to be in the rebuild phase. After the target image is restored, any additional table space restores that occur will restore data into existing table spaces, as defined in the rebuilt database. These table spaces will then be rolled forward with the database at the completion of the rebuild operation.

If you issue the **RESTORE** command with the appropriate **REBUILD** option and the database does not exist, a new database is created based on the attributes in the target image. If the database does exist, you will receive a warning message notifying you that the rebuild phase is starting. You will be asked if you want to continue the rebuild operation or not.

The rebuild operation restores all initial metadata from the target image. This includes all data that belongs to the database and does not belong to the table space data or the log files. Examples of initial metadata are:

- Table spaces definitions
- The history file, which is a database file that records administrative operations

The rebuild operation also restores the database configuration. The target image sets the log chain that determines what images can be used for the remaining restores during the rebuild phase. Only images on the same log chain can be used.

If a database already exists on disk and you want the history file to come from the target image, then you should specify the **REPLACE HISTORY FILE** option. The history file on disk at this time is used by the automatic logic to find the remaining images needed to rebuild the database.

Once the target image is restored:

- if the database is recoverable, the database is put into rollforward pending state and all table spaces that you restore are also put into rollforward pending state. Any table spaces defined in the database but not restored are put in restore pending state.
- If the database is not recoverable, then the database and the table spaces restored will go into normal state. Any table spaces not restored are put in drop pending state, as they can no longer be recovered. For this type of database, the rebuild phase is complete.

For recoverable databases, the rebuild phase ends when the first **ROLLFORWARD DATABASE** command is issued and the rollforward utility begins processing log records. If a rollforward operation fails after starting to process log records and a restore operation is issued next, the restore is not considered to be part of the rebuild phase. Such restores should be considered as normal table space restores that are not part of the rebuild phase.

Automatic processing

After the target image is restored, the restore utility determines if there are remaining table spaces that need to be restored. If there are, they are restored using the same connection that was used for running the **RESTORE DATABASE** command with the **REBUILD** option. The utility uses the history file on disk to find the most recent backup images taken prior to the target image that contains each of the remaining table spaces that needs to be restored. The restore utility uses the backup image location data stored in the history file to restore each of these images automatically. These subsequent restores, which are table space level restores, can be performed only offline. If the image selected does not belong on the current log chain, an error is returned. Each table space that is restored from that image is placed in rollforward pending state.

The restore utility tries to restore all required table spaces automatically. In some cases, it will not be able to restore some table spaces due to problems with the history file, or an error will occur restoring one of the required images. In such a case, you can either finish the rebuild manually or correct the problem and reissue the rebuild.

If automatic rebuilding cannot complete successfully, the restore utility writes to the diagnostics log (**db2diag** log file) any information it gathered for the remaining restore steps. You can use this information to complete the rebuild manually.

If a database is being rebuilt, only containers belonging to table spaces that are part of the rebuild process will be acquired.

If any containers need to be redefined through redirected restore, you will need to set the new path and size of the new container for the remaining restores and the subsequent rollforward operation.

If the data for a table space restored from one of these remaining images cannot fit into the new container definitions, the table space is put into restore pending state and a warning message is returned at the end of the restore. You can find additional information about the problem in the diagnostic log.

Completing the rebuild phase

Once all the intended table spaces have been restored you have two options based on the configuration of the database. If the database is nonrecoverable, the database will be connectable and any table spaces restored will be online. Any table spaces that are in drop pending state can no longer be recovered and should be dropped if future backups will be performed on the database.

If the database is recoverable, you can issue the rollforward command to bring the table spaces that were restored online. If SYSCATSPACE has not been restored, the rollforward will fail and this table space will have to be restored before the rollforward operation can begin. This means that during the rebuild phase, SYSCATSPACE must be restored.

Note: In a partitioned database environment, SYSCATSPACE does not exist on non-catalog partitions so it cannot be rebuilt there. However, on the catalog partition, SYSCATSPACE must be one of the table spaces that is rebuilt, or the rollforward operation will not succeed.

Rolling the database forward brings the database out of rollforward pending state and rolls any table spaces in rollforward pending state forward. The rollforward utility will not operate on any table space in restore pending state.

The stop time for the rollforward operation must be a time that is later than the end time of the most recent backup image restored during the rebuild phase. An error will occur if any other time is given. If the rollforward operation is not able to reach the backup time of the oldest image that was restored, the rollforward utility will not be able to bring the database up to a consistent point, and the rollforward fails.

You must have all log files for the time frame between the earliest and most recent backup images available for the rollforward utility to use. The logs required are those logs which follow the log chain from the earliest backup image to the target backup image, as defined by the truncation array in the target image, otherwise the rollforward operation will fail. If any backup images more recent than the target image were restored during the rebuild phase, then the additional logs from the target image to the most recent backup image restored are required. If the logs are not made available, the rollforward operation will put those table spaces that were not reached by the logs into restore pending state. You can issue the **LIST HISTORY** command to show the restore rebuild entry with the log range that will be required by roll forward.

The correct log files must be available. If you rely on the rollforward utility to retrieve the logs, you must ensure that the DB2 Log Manager is configured to indicate the location from which log files can be retrieved. If the log path or archive path has changed, you need to use the **OVERFLOW LOG PATH** option of the **ROLLFORWARD DATABASE** command.

Use the **AND STOP** option of the **ROLLFORWARD DATABASE** command to make the database available when the rollforward command successfully completes. At this point, the database is no longer in rollforward pending state. If the rollforward operation begins, but an error occurs before it successfully completes, the rollforward operation stops at the point of the failure and an error is returned. The database remains in rollforward pending state. You must take steps to correct the problem (for example, fix the log file) and then issue another rollforward operation to continue processing.

If the error cannot be fixed, you will be able to bring the database up at the point of the failure by issuing the **ROLLFORWARD STOP** command. Any log data beyond that point in the logs will no longer be available once the **STOP** option is used. The database comes up at that point and any table spaces that have been recovered are online. Table spaces that have not yet been recovered are in restore pending state. The database is in the normal state.

You will have to decide what is the best way to recover the remaining table spaces in restore pending state. This could be by doing a new restore and roll forward of a table space or by reissuing the whole rebuild operation again. This will depend on the type of problems encountered. In the situation where SYSCATSPACE is one of the table spaces in restore pending state, the database will not be connectable.

Database rebuild and table space containers

During a database rebuild, only those table spaces that are part of the rebuild process have their containers acquired. The containers belonging to each table space are acquired at the time the table space user data is restored out of an image.

When the target image is restored, each table space known to the database at the time of the backup has its definitions restored. This means the database created by the rebuild has knowledge of the same table spaces it did at backup time. For those table spaces that should also have their user data restored from the target image, their containers are also be acquired at this time.

Any remaining table spaces that are restored through intermediate table space restores have their containers acquired at the time the image that contains the table space data is restored.

Rebuild with redirected restore

In the case of redirected restore, all table space containers must be defined during the restore of the target image. If you specify the **REDIRECT** option, control is given back to you to redefine your table space containers. If you have redefined table space containers using the **SET TABLESPACE CONTAINERS** command then those new containers are acquired at that time. Any table space containers that you have not redefined are acquired as normal, at the time the table space user data is restored out of an image.

If the data for a table space that is restored cannot fit into the new container definitions, the table space is put into restore-pending state and a warning (SQL2563W) is returned to you at the end of the restore. There will also be a message in the DB2 diagnostics log detailing the problem.

Database rebuild and temporary table spaces

Temporary table spaces are stored differently than other database components in a backup image. Because they are stored differently, temporary table spaces are rebuilt differently during a database restoration.

In general, a DB2 backup image is made up of the following components:

- Initial database metadata, such as the table space definitions, database configuration file, and history file.
- Data for non-temporary table spaces specified to the **BACKUP** utility
- Final database metadata such as the log file header
- Log files (if the **INCLUDE LOGS** option was specified)

In every backup image, whether it is a database or table space backup, a full or incremental (delta) backup, these core components can always be found.

A database backup image will contain all of the previously listed components, as well as data for every table space defined in the database at the time of the backup.

A table space backup image will always include the database metadata listed previously, but it will only contain data for those table spaces that are specified to the backup utility.

Temporary table spaces are treated differently than nontemporary table spaces. Temporary table space data is never backed up, but their existence is important to the framework of the database. Although temporary table space data is never backed up, the temporary table spaces are considered part of the database, so they are specially marked in the metadata that is stored with a backup image. This makes it look like they are in the backup image. In addition, the table space definitions hold information about the existence of any temporary table spaces.

Although no backup image ever contains data for a temporary table space, during a database rebuild operation when the target image is restored (regardless the type of image), temporary table spaces are also restored, only in the sense that their containers are acquired and allocated. The acquisition and allocation of containers is done automatically as part of the rebuild processing. As a result, when rebuilding a database, you cannot exclude temporary table spaces.

Choosing a target image for database rebuild

The rebuild target image should be the most recent backup image that you want to use as the starting point of your restore operation.

This image is known as the target image of the rebuild operation, because it defines the structure of the database to be restored, including the table spaces that can be restored, the database configuration, and the log sequence.It can be any type of backup (full, table space, incremental, online or offline).

The target image sets the log sequence (or log chain) that determines what images can be used for the remaining restores during the rebuild phase. Only images on the same log chain can be used.

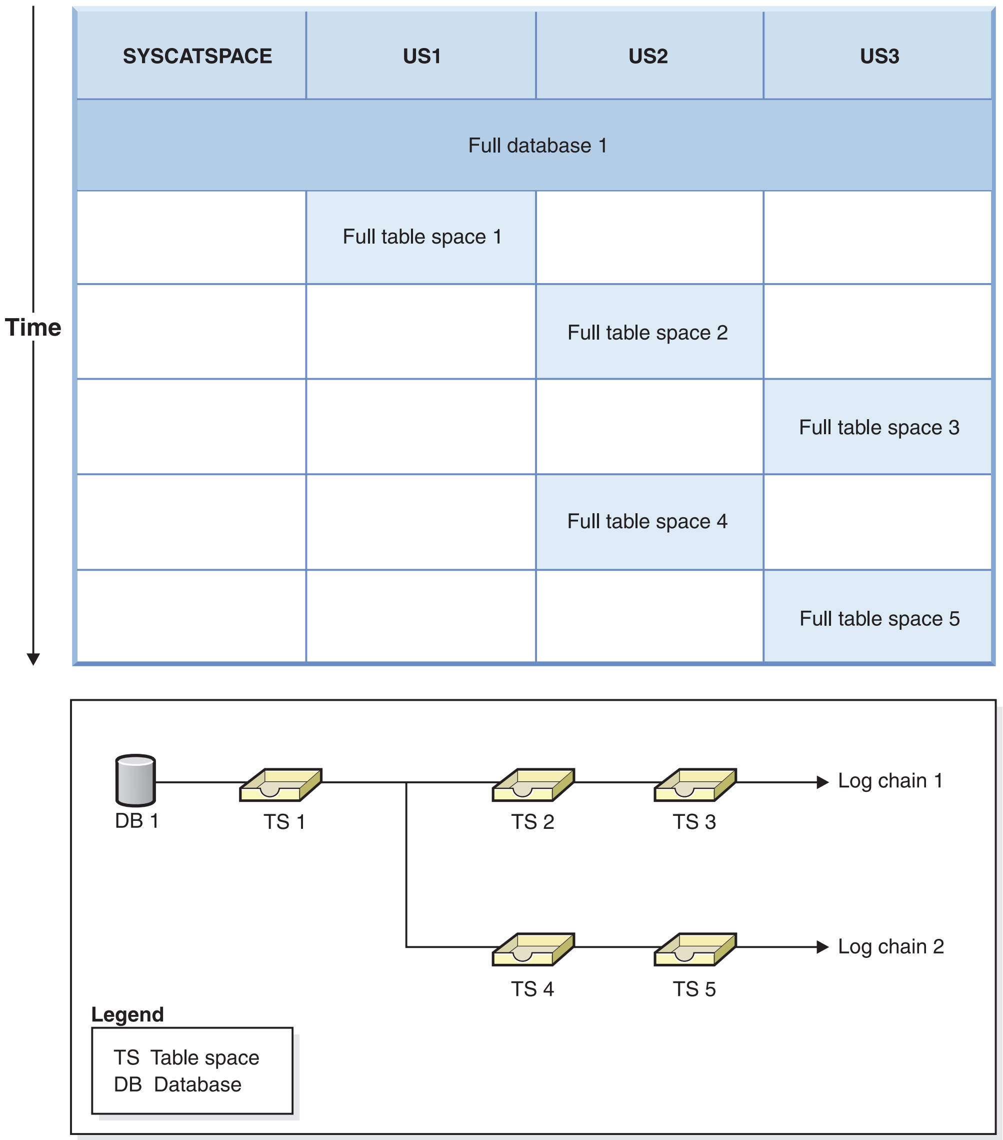
The following examples illustrate how to choose the image you should use as the target image for a rebuild operation.

Suppose there is a database called SAMPLE that has the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

Figure 59 on page 808 shows that the following database-level backups and table space-level backups that have been taken, in chronological order:

1. Full database backup DB1
2. Full table space backup TS1
3. Full table space backup TS2
4. Full table space backup TS3
5. Database restore and roll forward to a point between TS1 and TS2
6. Full table space backup TS4
7. Full table space backup TS5



SYSCATSPACE	US1	US2	US3
Full database 1			
	Full table space 1		
		Full table space 2	
			Full table space 3
		Full table space 4	
			Full table space 5

Figure 59. Database and table space-level backups of database SAMPLE

Example 1

The following example demonstrates the CLP commands you need to issue to rebuild database SAMPLE to the current point of time. First you need to choose the table spaces you want to rebuild. Since your goal is to rebuild the database to the current point of time you need to select the most recent backup image as your target image. The most recent backup image is image TS5, which is on log chain 2:

```
db2 restore db sample rebuild with all tablespaces in database taken at
   TS5 without prompting
db2 rollforward db sample to end of logs
db2 rollforward db sample stop
```

This restores backup images TS5, TS4, TS1 and DB1 automatically, then rolls the database forward to the end of log chain 2.

Note: All logs belonging to log chain 2 must be accessible for the rollforward operations to complete.

Example 2

This second example demonstrates the CLP commands you need to issue to rebuild database SAMPLE to the end of log chain 1. The target image you select should be the most recent backup image on log chain 1, which is TS3:

```
db2 restore db sample rebuild with all tablespaces in database
   taken at TS3 without prompting
db2 rollforward db sample to end of logs
db2 rollforward db sample stop
```

This restores backup images TS3, TS2, TS1, and DB1 automatically, then rolls the database forward to the end of log chain 1.

Note:

- All logs belonging to log chain 1 must be accessible for the rollforward operations to complete.
- This command may fail because a log file is retrieved from a higher log chain (the rollforward utility always attempts to get log files from the highest log chain), you need to do the following steps:
  1. Extract the log files manually to the overflow log path.
  2. Run the **ROLLFORWARD** command. Include the parameters **-OVERFLOW LOG PATH**, to specify the location of the extracted log files, and **-NORETRIEVE**, to disable the retrieval of archived logs.

Choosing the wrong target image for rebuild

Suppose there is a database called SAMPLE2 that has the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)

Figure 60 on page 810 shows the backup log chain for SAMPLE2, which consists of the following backups:

1. BK1 is a full database backup, which includes all table spaces
2. BK2 is a full table space backup of USERSP1
3. BK3 is a full table space backup of USERSP2

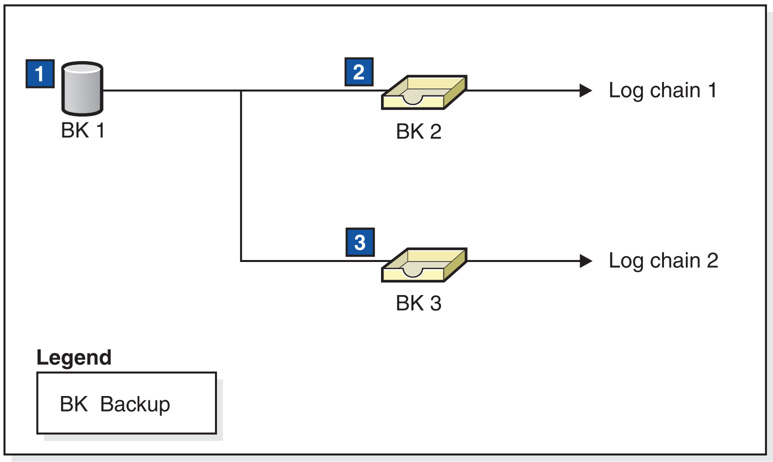


Figure 60. Backup log chain for database SAMPLE2

The following example demonstrates the CLP command you need to issue to rebuild the database from BK3 using table spaces SYSCATSPACE and USERSP2:

```
db2 restore db sample2 rebuild with tablespace (SYSCATSPACE,
   USERSP2) taken at BK3 without prompting
```

Now suppose that after this restore completes, you decide that you also want to restore USERSP1, so you issue the following command:

```
db2 restore db sample2 tablespace (USERSP1) taken at BK2
```

This restore fails and provides a message that says BK2 is from the wrong log chain (SQL2154N). As you can see in Figure 60, the only image that can be used to restore USERSP1 is BK1. Therefore, you need to type the following command:

```
db2 restore db sample2 tablespace (USERSP1) taken at BK1
```

This succeeds so that database can be rolled forward accordingly.

Rebuilding selected table spaces

Rebuilding a database allows you to build a database that contains a subset of the table spaces that make up the original database.

About this task

Rebuilding only a subset of table spaces within a database can be useful in the following situations:

- In a test and development environment in which you want to work on only a subset of table spaces.
- In a recovery situation in which you need to bring table spaces that are more critical online faster than others, you can first restore a subset of table spaces then restore other table spaces at a later time.

To rebuild a database that contains a subset of the table spaces that make up the original database, consider the following example.

In this example, there is a database named SAMPLE that has the following table spaces:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)

- USERSP2 (user data table space)
- USERSP3 (user data table space)

Figure 61 shows that the following backups have been taken:

- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

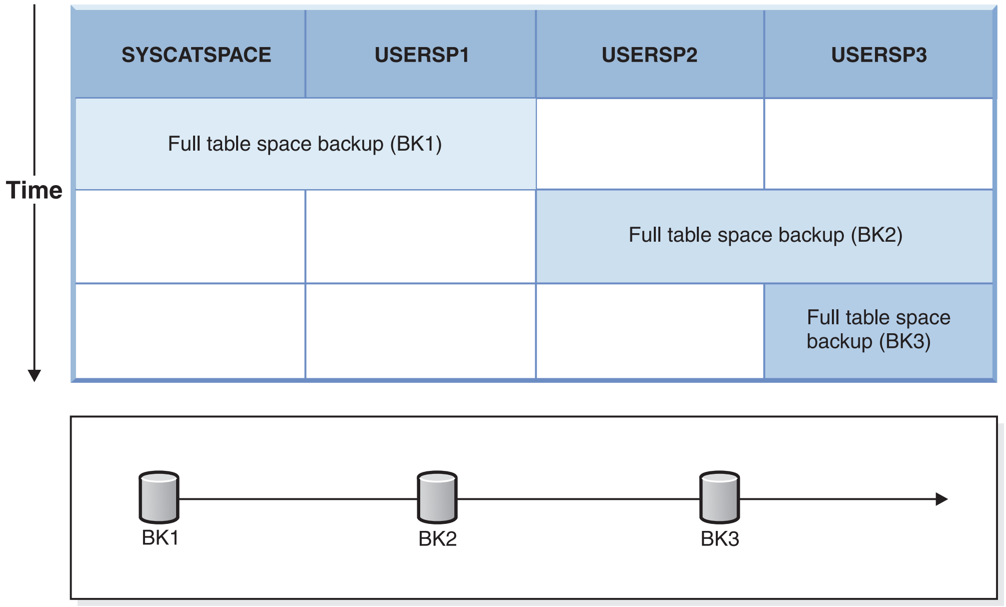

Time
SYSCATSPACE
USERSP1
USERSP2
USERSP3
Full table space backup (BK1)
Full table space backup (BK2)
Full table space backup (BK3)
BK1
BK2
BK3

Figure 61. Backup images available for database SAMPLE

The following procedure demonstrates using the **RESTORE DATABASE** and **ROLLFORWARD DATABASE** commands, issued through the CLP, to rebuild just SYSCATSPACE and USERSP1 to end of logs:

```
db2 restore db mydb rebuild with all tablespaces in image
        taken at BK1 without prompting
db2 rollforward db mydb to end of logs
db2 rollforward db mydb stop
```

At this point the database is connectable and only SYSCATSPACE and USERSP1 are in NORMAL state. USERSP2 and USERSP3 are in restore-pending state. You can still restore USERSP2 and USERSP3 at a later time.

Rebuild and incremental backup images

You can rebuild a database using incremental images.

By default, the restore utility tries to use automatic incremental restore for all incremental images. This means that if you do not use the **INCREMENTAL** option of the **RESTORE DATABASE** command, but the target image is an incremental backup image, the restore utility will issue the rebuild operation using automatic incremental restore. If the target image is not an incremental image, but another required image is an incremental image then the restore utility will make sure those incremental images are restored using automatic incremental restore. The restore utility will behave in the same way whether you specify the **INCREMENTAL** option with the **AUTOMATIC** option or not.

If you specify the **INCREMENTAL** option but not the **AUTOMATIC** option, you will need to perform the entire rebuild process manually. The restore utility will just restore the initial metadata from the target image, as it would in a regular manual incremental restore. You will then need to complete the restore of the target image using the required incremental restore chain. Then you will need to restore the remaining images to rebuild the database.

It is recommended that you use automatic incremental restore to rebuild your database. Only in the event of a restore failure, should you attempt to rebuild a database using manual methods.

Rebuilding partitioned databases

To rebuild a partitioned database, rebuild each database partition separately. For each database partition, beginning with the catalog partition, first restore all the table spaces that you require. Any table spaces that are not restored are placed in restore pending state.

Once all the database partitions are restored, you then issue the **ROLLFORWARD DATABASE** command on the catalog partition to roll all of the database partitions forward.

About this task

Note: If, at a later date, you need to restore any table spaces that were not originally included in the rebuild phase, you need to make sure that when you subsequently roll the table space forward that the rollforward utility keeps all the data across the database partitions synchronized. If a table space is missed during the original restore and rollforward operation, it might not be detected until there is an attempt to access the data and a data access error occurs. You will then need to restore and roll the missing table space forward to get it back in sync with the rest of the partitions.

To rebuild a partitioned database using table space level backup images, consider the following example.

In this example, there is a recoverable database called SAMPLE with three database partitions:

- Database partition 1 contains table spaces SYSCATSPACE, USERSP1 and USERSP2, and is the catalog partition
- Database partition 2 contains table spaces USERSP1 and USERSP3
- Database partition 3 contains table spaces USERSP1, USERSP2 and USERSP3

The following backups have been taken, where BK*xy* represents backup number *x* on partition *y*:

- BK11 is a backup of SYSCATSPACE, USERSP1 and USERSP2
- BK12 is a backup of USERSP2 and USERSP3
- BK13 is a backup of USERSP1, USERSP2 and USERSP3
- BK21 is a backup of USERSP1
- BK22 is a backup of USERSP1
- BK23 is a backup of USERSP1
- BK31 is a backup of USERSP2
- BK33 is a backup of USERSP2

- BK42 is a backup of USERSP3
- BK43 is a backup of USERSP3

The following procedure demonstrates using the **RESTORE DATABASE** and **ROLLFORWARD DATABASE** commands, issued through the CLP, to rebuild the entire database to the end of logs.

Procedure

1. On database partition 1, issue a **RESTORE DATABASE** command with the **REBUILD** option:
   ```
   db2 restore db sample rebuild with all tablespaces in database
      taken at BK31 without prompting
   ```
2. On database partition 2, issue a **RESTORE DATABASE** command with the **REBUILD** option:
   ```
   db2 restore db sample rebuild with tablespaces in database
      taken at BK42 without prompting
   ```
3. On database partition 3, issue a **RESTORE DATABASE** command with the **REBUILD** option:
   ```
   db2 restore db sample rebuild with all tablespaces in database
      taken at BK43 without prompting
   ```
4. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option:
   ```
   db2 rollforward db sample to end of logs
   ```
5. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:
   ```
   db2 rollforward db sample stop
   ```

What to do next

At this point the database is connectable on all database partitions and all table spaces are in NORMAL state.

Restrictions for database rebuild

You can use the REBUILD option to complete a set of restore commands, but it has restrictions that you need to be aware of.

The following list is a summary of database rebuild restrictions:

- One of the table spaces you rebuild must be SYSCATSPACE on the catalog partition.
- You must either issue commands using the command line processor (CLP) or use the corresponding application programming interfaces (APIs) to perform a rebuild operation.
- The REBUILD option cannot be used against a pre-Version 9.1 target image unless the image is that of an offline database backup. If the target image is an offline database backup, then only the table spaces in this image can be used for the rebuild. The database needs to be migrated after the rebuild operation successfully completes. Attempts to rebuild using any other type of pre-Version 9.1 target image result in an error.
- The REBUILD option cannot be issued against a target image from a different operating system than the one being restored on unless the target image is a full database backup. If the target image is a full database backup, then only the table spaces in this image can be used for the rebuild. Attempts to rebuild using any other type of target image from a different operating system than the one being restored on result in an error.

- The TRANSPORT option is incompatible with the REBUILD option.

Rebuild sessions - CLP examples

This topic provides a number of examples of rebuild operations.

Scenario 1

In the following examples, there is a recoverable database called MYDB with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:

- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

Example 1

The following rebuilds the entire database to the most recent point in time:

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:
   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK3 without prompting
   ```
2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to end of logs
   ```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 2

The following rebuilds just SYSCATSPACE and USERSP2 to a point in time (where end of BK3 is less recent than the point in time, which is less recent than end of logs):

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option and specify the table spaces you want to include.
   ```
   db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
      taken at BK2 without prompting
   ```
2. Issue a **ROLLFORWARD DATABASE** command with the **TO PIT** option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to PIT
   ```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 and USERSP3 are in RESTORE_PENDING state.

To restore USERSP1 and USERSP3 at a later time, using normal table space restores (without the **REBUILD** option):

1. Issue the **RESTORE DATABASE** command *without* the **REBUILD** option and specify the table space you want to restore. First restore USERSPI:

   ```
   db2 restore db mydb tablespace (USERSP1) taken at BK1 without prompting
   ```
2. Then restore USERSP3:

   ```
   db2 restore db mydb tablespace taken at BK3 without prompting
   ```
3. Issue a **ROLLFORWARD DATABASE** command with the **END OF LOGS** option and specify the table spaces to be restored (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs tablespace (USERSP1, USERSP3)
   ```

   The rollforward will replay all logs up to the PIT and then stop for these two table spaces since no work has been done on them since the first rollforward.
4. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

Example 3

The following rebuilds just SYSCATSPACE and USERSP1 to end of logs:

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:

   ```
   db2 restore db mydb rebuild with all tablespaces in image
      taken at BK1 without prompting
   ```
2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and only SYSCATSPACE and USERSP1 are in NORMAL state. USERSP2 and USERSP3 are in RESTORE_PENDING state.

Example 4

In the following example, the backups BK1 and BK2 are no longer in the same location as stated in the history file, but this is not known when the rebuild is issued.

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option, specifying that you want to rebuild the entire database to the most recent point in time:

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK3 without prompting
   ```

   At this point, the target image is restored successfully, but an error is returned from the restore utility stating it could not find a required image.
2. You must now complete the rebuild manually. Since the database is in the rebuild phase this can be done as follows:
   a. Issue a **RESTORE DATABASE** command and specify the location of the BK1 backup image:

      ```
      db2 restore db mydb tablespace taken at BK1 from location
         without prompting
      ```
   b. Issue a **RESTORE DATABASE** command and specify the location of the BK2 backup image:

      ```
      db2 restore db mydb tablespace (USERSP2) taken at BK2 from
         location without prompting
      ```

c. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```

d. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 5

In this example, table space USERSP3 contains independent data that is needed for generating a specific report, but you do not want the report generation to interfere with the original database. In order to gain access to the data but not affect the original database, you can use **REBUILD** to generate a new database with just this table space and SYSCATSPACE. SYSCATSPACE is also required so that the database will be connectable after the restore and roll forward operations.

To build a new database with the most recent data in SYSCATSPACE and USERSP3:

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option, and specify that table spaces SYSCATSPACE and USERSP3 are to be restored to a new database, NEWDB:

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP3)
   taken at BK3 into newdb without prompting
```

2. Issue a **ROLLFORWARD DATABASE** command on NEWDB with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db newdb to end of logs
```

3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db newdb stop
```

At this point the new database is connectable and only SYSCATSPACE and USERSP3 are in NORMAL state. USERSP1 and USERSP2 are in RESTORE_PENDING state.

Note: If container paths are an issue between the current database and the new database (for example, if the containers for the original database need to be altered because the file system does not exist or if the containers are already in use by the original database) then you will need to perform a redirected restore. This example assumes the default autostorage database paths are used for the table spaces.

Scenario 2

In the following example, there is a recoverable database called MYDB that has SYSCATSPACE and one thousand user table spaces named T*xxxx*, where *xxxx* stands for the table space number (for example, T0001). There is one full database backup image (BK1)

Example 6

The following restores all table spaces except T0999 and T1000:

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in image except
   tablespace (T0999, T1000) taken at BK1 without prompting
```

2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```

3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database will be connectable and all table spaces except T0999 and T1000 will be in NORMAL state. T0999 and T1000 will be in RESTORE_PENDING state.

Scenario 3

The examples in this scenario demonstrate how to rebuild a recoverable database using incremental backups. In the following examples, there is a database called MYDB with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:

- FULL1 is a full backup of SYSCATSPACE, USERSP1, USERSP2 and USERSP3
- DELTA1 is a delta backup of SYSCATSPACE and USERSP1
- INCR1 is an incremental backup of USERSP2 and USERSP3
- DELTA2 is a delta backup of SYSCATSPACE, USERSP1, USERSP2 and USERSP3
- DELTA3 is a delta backup of USERSP2
- FULL2 is a full backup of USERSP1

Example 7

The following rebuilds just SYSCATSPACE and USERSP2 to the most recent point in time using incremental automatic restore.

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option. The **INCREMENTAL AUTO** option is optional. The restore utility will detect what the granularity of the image is and use automatic incremental restore if it is required.

   ```
   db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
      incremental auto taken at DELTA3 without prompting
   ```

2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```

3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 and USERSP3 are in RESTORE_PENDING state.

Example 8

The following rebuilds the entire database to the most recent point in time using incremental automatic restore.

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option. The **INCREMENTAL AUTO** option is optional. The restore utility will detect what the granularity of the image is and use automatic incremental restore if it is required.

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      incremental auto taken at DELTA3 without prompting
   ```
2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 9

The following rebuilds the entire database, except for USERSP3, to the most recent point in time.

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option. Although the target image is a non-incremental image, the restore utility will detect that the required rebuild chain includes incremental images and it will automatically restore those images incrementally.

   ```
   db2 restore db mydb rebuild with all tablespaces in database except
      tablespace (USERSP3) taken at FULL2 without prompting
   ```
2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

Scenario 4

The examples in this scenario demonstrate how to rebuild a recoverable database using backup images that contain log files. In the following examples, there is a database called MYDB with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)

Example 10

The following rebuilds the database with just SYSCATSPACE and USERSP2 to the most recent point in time. There is a full online database backup image (BK1), which includes log files.

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:

   ```
   db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
      taken at BK1 logtarget /logs without prompting
   ```
2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs after the end of BK1 have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs overflow log path (/logs)
   ```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 is in RESTORE_PENDING state.

Example 11

The following rebuilds the database to the most recent point in time. There are two full online table space backup images that include log files:

- BK1 is a backup of SYSCATSPACE, using log files 10-45
- BK2 is a backup of USERSP1 and USERSP2, using log files 64-80

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK2 logtarget /logs without prompting
   ```

   The rollforward operation will start at log file 10, which it will always find in the overflow log path if not in the primary log file path. The log range 46-63, since they are not contained in any backup image, will need to be made available for roll forward.
2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option, using the overflow log path for log files 64-80:

   ```
   db2 rollforward db mydb to end of logs overflow log path (/logs)
   ```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Scenario 5

In the following examples, there is a recoverable database called MYDB with the following table spaces in it:

- SYSCATSPACE (0), SMS system catalog (relative container)
- USERSP1 (1) DMS user data table space (absolute container /usersp2)
- USERSP2 (2) DMS user data table space (absolute container /usersp3)

The following backups have been taken:

- BK1 is a backup of SYSCATSPACE
- BK2 is a backup of USERSP1 and USERSP2
- BK3 is a backup of USERSP2

Example 12

The following rebuilds the entire database to the most recent point in time using redirected restore.

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK3 redirect without prompting
   ```
2. Issue a **SET TABLESPACE CONTAINERS** command for each table space whose containers you want to redefine. For example:

   ```
   db2 set tablespace containers for 3 using (file '/newusersp1' 10000)
   ```
3.

   ```
   db2 set tablespace containers for 4 using (file '/newusersp2' 15000)
   ```
4. Issue a **RESTORE DATABASE** command with the **CONTINUE** option:

   ```
   db2 restore db mydb continue
   ```

5. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

   ```
   db2 rollforward db mydb to end of logs
   ```
6. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Scenario 6

In the following examples, there is a database called MYDB with three database partitions:

- Database partition 1 contains table spaces SYSCATSPACE, USERSP1 and USERSP2, and is the catalog partition
- Database partition 2 contains table spaces USERSP1 and USERSP3
- Database partition 3 contains table spaces USERSP1, USERSP2 and USERSP3

The following backups have been taken, where BK*xy* represents backup number *x* on partition *y*:

- BK11 is a backup of SYSCATSPACE, USERSP1 and USERSP2
- BK12 is a backup of USERSP2 and USERSP3
- BK13 is a backup of USERSP1, USERSP2 and USERSP3
- BK21 is a backup of USERSP1
- BK22 is a backup of USERSP1
- BK23 is a backup of USERSP1
- BK31 is a backup of USERSP2
- BK33 is a backup of USERSP2
- BK42 is a backup of USERSP3
- BK43 is a backup of USERSP3

Example 13

The following rebuilds the entire database to the end of logs.

1. On database partition 1, issue a **RESTORE DATABASE** command with the **REBUILD** option:

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK31 without prompting
   ```
2. On database partition 2, issue a **RESTORE DATABASE** command with the **REBUILD** option:

   ```
   db2 restore db mydb rebuild with tablespaces in database taken at
      BK42 without prompting
   ```
3. On database partition 3, issue a **RESTORE DATABASE** command with the **REBUILD** option:

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK43 without prompting
   ```
4. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (assumes all logs have been saved and are accessible on all database partitions):

   ```
   db2 rollforward db mydb to end of logs
   ```
5. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable on all database partitions and all table spaces are in NORMAL state.

Example 14

The following rebuilds SYSCATSPACE, USERSP1 and USERSP2 to the most recent point in time.

1. On database partition 1, issue a **RESTORE DATABASE** command with the **REBUILD** option:

   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK31 without prompting
   ```
2. On database partition 2, issue a **RESTORE DATABASE** command with the **REBUILD** option:

   ```
   db2 restore db mydb rebuild with all tablespaces in image taken at
      BK22 without prompting
   ```
3. On database partition 3, issue a **RESTORE DATABASE** command with the **REBUILD** option:

   ```
   db2 restore db mydb rebuild with all tablespaces in image taken at
      BK33 without prompting
   ```

   Note: this command omitted USERSP1, which is needed to complete the rebuild operation.
4. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option:

   ```
   db2 rollforward db mydb to end of logs
   ```
5. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

   ```
   db2 rollforward db mydb stop
   ```

   The rollforward succeeds and the database is connectable on all database partitions. All table spaces are in NORMAL state, except USERSP3, which is in RESTORE PENDING state on all database partitions on which it exists, and USERSP1, which is in RESTORE PENDING state on database partition 3.

   When an attempt is made to access data in USERSP1 on database partition 3, a data access error will occur. To fix this, USERSP1 will need to be recovered:

   a. On database partitions 3, issue a **RESTORE DATABASE** command, specifying a backup image that contains USERSP1:

      ```
      db2 restore db mydb tablespace taken at BK23 without prompting
      ```
   b. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option and the **AND STOP** option:

      ```
      db2 rollforward db mydb to end of logs on dbpartitionnum (3) and stop
      ```

At this point USERSP1 on database partition 3 can have its data accessed since it is in NORMAL state.

Scenario 7

In the following examples, there is a *nonrecoverable* database called MYDB with the following table spaces:

- SYSCATSPACE (0), SMS system catalog
- USERSP1 (1) DMS user data table space
- USERSP2 (2) DMS user data table space

There is just one backup of the database, BK1:

Example 15

The following demonstrates using rebuild on a nonrecoverable database.

Rebuild the database using only SYSCATSPACE and USERSP1:

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP1)
   taken at BK1 without prompting
```

Following the restore, the database is connectable. If you issue the **LIST TABLESPACES** command or the MON_GET_TABLESPACE table function, you see that the SYSCATSPACE and USERSP1 are in NORMAL state, while USERSP2 is in DELETE_PENDING/OFFLINE state. You can now work with the two table spaces that are in NORMAL state.

If you want to do a database backup, you will first need to drop USERSP2 using the DROP TABLESPACE statement, otherwise, the backup will fail.

To restore USERSP2 at a later time, you need to reissue a database restore from BK1.

Database schema transporting

Transporting a database schema involves taking a backup image of a database and restoring the database schema to a different, existing database.

When you transport a database schema, the database objects in the transported schema are re-created to reference the new database, and the data is restored to the new database.

A database schema must be transported in its entirety. If a table space contains both the schema you want to transport, as well as another schema, you must transport all data objects from both schemas. These sets of schemas that have no references to other database schemas are called *transportable sets*. The data in the table spaces and logical objects in the schemas in a transportable set reference only table spaces and schemas in the transportable set. For example, tables have table dependencies only on other tables in the transportable set.

The following diagram illustrates a database with several table spaces and schemas. In the diagram, the table spaces that are referenced by the schemas are above the schemas. Some schemas reference multiple table spaces and some table spaces are referenced by multiple schemas.

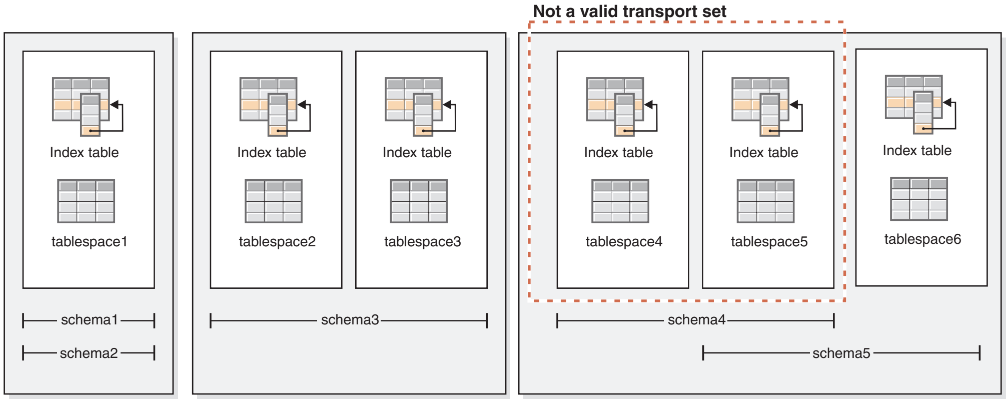


Figure 62. Sets of table spaces and schemas

The following combinations of table spaces and schemas are valid transportable sets:

- tablespace1 with schema1 and schema2
- tablespace2 and tablespace3 with schema3
- tablespace4, tablespace5, and tablespace6, with schema4 and schema5
- A combination of valid transportable sets also constitutes a valid transportable set:
  - tablespace1, tablespace2, and tablespace3, with schema1, schema2, and schema3

The set tablespace4 and tablespace5 with schema4 is not a valid transportable set because there are references between tablespace5 and schema5 and between schema5 and tablespace6. The set requires tablespace6 with schema5 to be a valid transportable set.

You can transport database schemas by using the **RESTORE** command with the **TRANSPORT** parameter.

When you transport database schemas, a temporary database is created and named as a part of the transport operation. This *transport staging database* is used to extract logical objects from the backup image so that they can be re-created on the target database. If logs are included in the backup image, they are also used to bring the staging database to a point of transactional consistency. The ownership of the transported table spaces is then transferred to the target database.

Considerations about the database objects re-created when transporting database schemas

Review the following information related to the re-creation of database objects when you are transporting database schemas:

Table 127. Transport considerations for specific database objects

Database object	Consideration when transporting schemas
SQL routines (not external routines using SQL)	A new copy of the SQL routine is created in the target database. For SQL stored procedures, additional catalog space is consumed because an additional copy of the stored procedure byte code is created in the new database.

Table 127. Transport considerations for specific database objects (continued)

Database object	Consideration when transporting schemas
External routines	A new catalog entry is created for each routine. This catalog entry references the same binary file as the original source routine. The **RESTORE** command does not copy the external routine binary file from the source system.
Source tables in states causing access problems	For tables that are not in normal state at the time the backup image was generated, such as tables in check pending state or load pending state, the data from those tables might not be accessible in the target database. To avoid having this inaccessible data, you can move the tables to normal state in the source database before schema transport.
Tables containing the data capture attribute	Source tables with data capture enabled are transported to the target database with the data capture attribute and continue to log interdatabase data replication information. However, replicated tables do not extract information from this table. You have the option of registering the new target table to act as a replication source after the **RESTORE** command completes.
Tables using label-based access control (LBAC)	When transporting data that is protected by LBAC, the transport operation re-creates the LBAC objects on the target database. If LBAC objects of the same name exist on the target database, the transport operation fails. To ensure that restricted data access is not compromised, the transport operation does not use existing LBAC objects on the target database.
Temporary table spaces	If there are any system temporary table spaces that are defined with the source backup image and the transport operation excludes them from the table space list, these system temporary table spaces are still created in the staging database but not the final target database. As a result, you must issue the **SET TABLESPACE CONTAINERS** command for these system temporary table spaces in order to provide valid containers to complete the restore operation, just as you would for any table spaces that are specified within the table space list.

When you transport table spaces, a log record with a special format is created on the target database. This format cannot be read by previous DB2 versions. If you transport table spaces and then downgrade to a version earlier than DB2 Version 9.7 Fix Pack 2, then you cannot recover the target database that contains the table spaces that were transported. To ensure that the target database is compatible with earlier DB2 versions, you can roll forward the target database to a point in time before the transport operation.

Important: If database rollforward detects a table space schema transport log record, the corresponding transported table space is taken offline and moved into drop pending state. This is because database does not have complete logs of transported table spaces to rebuild transported table spaces and their contents. You can take a full backup of the target database after transport completes, so subsequent rollforward does not pass the point of schema transport in the log stream.

Transportable objects

When you transport data from a backup image to a target database, there are two main results. The physical and logical objects in the table spaces that you are restoring are re-created in the target database, and the table space definitions and containers are added to the target database.

The following logical objects are re-created:

- Tables, created global temporary tables, and materialized query tables
- Normal and statistical views
- The following types of generated columns:
  - Expression
  - Identity
  - Row change timestamp
  - Row change token
- User-defined functions and generated functions
- Functions and procedures except for external routine executables
- User-defined types
- The following types of constraints:
  - Check
  - Foreign key
  - Functional dependency
  - Primary
  - Unique
- Indexes
- Triggers
- Sequences
- Object authorizations, privileges, security, access control, and audit configuration
- Table statistics, profiles, and hints
- Packages

The following components of a schema are not created on the target database:

- Aliases
- Created global variables
- External routine executable files
- Functional mappings and templates
- Hierarchy tables
- Index extensions
- Jobs
- Methods
- Nicknames
- OLE DB external functions
- Range-partitioned tables
- Servers
- Sourced procedures
- Structured types
- System catalogs
- Typed tables and typed views
- Usage lists
- Wrappers

Transport examples

You can use the **RESTORE DATABASE** command with the TRANSPORT option to copy a set of table spaces and SQL schemas from one database to another database.

The following examples use a database named ORIGINALDB as source of the backup image and the target database TARGETDB.

The following illustration shows the ORIGINALDB table spaces and schemas:

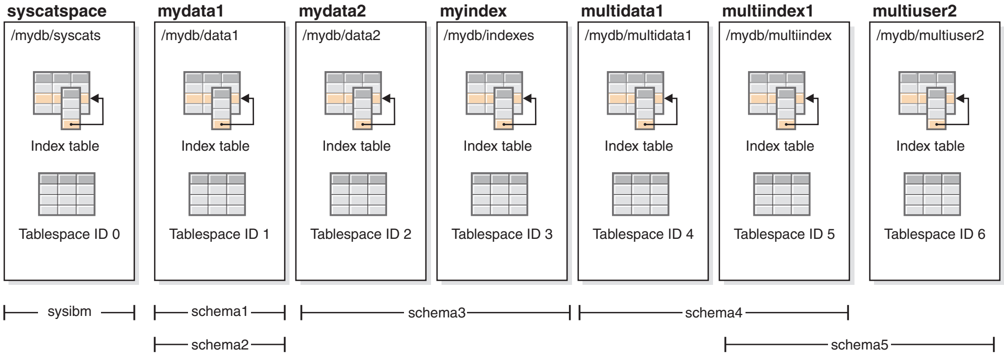


Figure 63. ORIGINALDB database

The originalDB database contains the following valid transportable sets:

- mydata1; schema1 + schema2
- mydata2 + myindex; schema3
- multidata1 + multiindex1 + multiuser2; schema4 + schema5
- A combination of valid transportable sets also constitutes a valid transportable set:
  - mydata1 + mydata2 + myindex; schema1 + schema + schema3

The following illustration shows the TARGETDB table spaces and schemas:

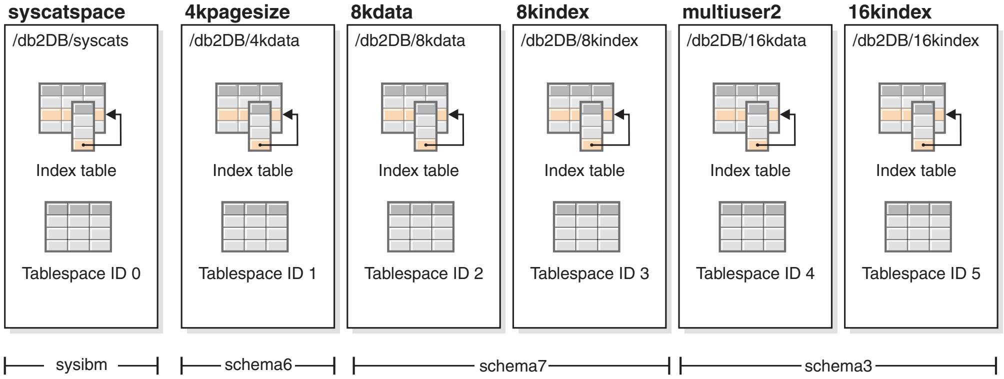


Figure 64. TARGETDB database

If the sources and target databases contain any schemas with the same schema name, or any table spaces of the table space name, then you cannot transport that schema or table space to the target database. Issuing a transport operation that contains a schema or a table space that has the same name as a schema or a table space on the target database will cause the transport operation to fail. For example, even though the following grouping is a valid transportable set, it cannot be directly transported to the target database:

- mydata2 + myindex; schema3 (schema3 exists in both the source and target databases)

If there exists a single online backup image for ORIGINALDB that contains all of the table spaces in the database, then this will be the source for the transport. This also applies to table space level backup images.

You can redirect the container paths for the table spaces being transported. This is especially important if database relative paths were used.

Examples

Example 1: Successfully transport the schemas schema1 and schema2 in the mydata1 table space into TARGETDB.

```
db2 restore db originaldb tablespace (mydata1) schema(schema1,schema2)
  from <Media_Target_clause> taken at <date-time>
  transport into targetdb redirect

db2 list tablespaces
db2 set tablespace containers for <tablespace ID for mydata1>
  using (path '/db2DB/data1')

db2 restore db originaldb continue
```

The resulting TARGETDB will contain the mydata1 table space and schema1 and schema2.

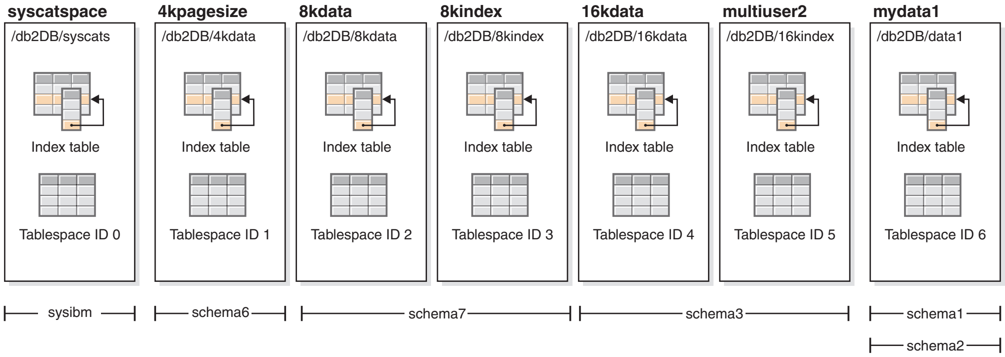


Figure 65. TARGETDB database after transport

Example 2: Transport the schema schema3 in the mydata2 and myindex table spaces into TARGETDB. You cannot transport a schema that already exists on the target database.

```
db2 restore db originaldb tablespace (mydata2,myindex) schema(schema3)
  transport into targetdb
```

The transport operation will fail because the schema schema3 already exists on the target database. TARGETDB will remain unchanged. SQLCODE=SQL2590N rc=3.

Example 3: Transport the schemas schema4 and schema5 in the multidata1, multiindex1, and multiuser2 table spaces into TARGETDB. You cannot transport a table space that already exists on the target database.

```
db2 restore db originaldb tablespace (multidata1,multiindex1,multiuser2)
  schema(schema4,schema5) transport into targetdb
```

The transport operation will fail and TARGETDB will remain unchanged because table space multiuser2 already exists on the target database. SQLCODE=SQL2590N rc=3.

Example 4: Transport the myindex table space into TARGETDB. You cannot transport partial schemas.

```
db2 restore db originaldb tablespace (myindex) schema(schema3)
  transport into targetdb
```

The list of table spaces and schemas being transported is not a valid transportable set. The transport operation will fail and TARGETDB will remain unchanged. SQLCODE=SQL2590N rc=1.

Example 5: Restore the syscatspace table space into TARGETDB. You cannot transport system catalogs.

```
db2 restore db originaldb tablespace (syscatspace) schema(sysibm)
  transport into targetdb
```

The transport operation will fail because the system catalogs can not be transported. SQLCODE=SQL2590N rc=4. You can transport user defined table spaces or restore the system catalogs with the RESTORE DATABASE command without specifying the transport option.

Example 6: You cannot restore into a target database that does not exist on the system.

```
db2 restore db originaldb tablespace (mydata1) schema(schema1,schema2)
  transport into notexists
```

The transport operation will fail. Table spaces cannot be transported to a target database that does not exist.

Troubleshooting: transporting schemas

If an error occurs on either the staging or target database, you must redo the entire restore operation. All failures that occur are logged in the `db2diag log` file on the target server. Review the **`db2diag`** log before reissuing the **`RESTORE`** command.

Dealing with errors

Errors occurring during restore are handled in various ways depending on the type of object being copied and the phase of transport. There might be circumstances, such as a power failure, in which not everything is cleaned up.

The transport operation consists of the following phases:

- Staging database creation
- Physical table space container restoration
- Rollforward processing
- Schema validation
- Transfer of ownership of the table space containers
- Schema re-creation in target database
- Dropping the staging database (if the **`STAGE IN`** parameter is not specified)

If any errors are logged at the end of the schema re-creation phase, about transporting physical objects, then the restore operation fails and an error is returned. All object creation on the target database is rolled back, and all internally created tables are cleaned up on the staging database. The rollback occurs at the end of the re-create phase, to allow all possible errors to be recorded into the **db2diag** log file. You can investigate all errors returned before reissuing the command.

The staging database is dropped automatically after success or failure. However, it is not dropped in the event of failure if the **STAGE IN** parameter is specified. The staging database must be dropped before the staging database name can be reused.

Monitoring the progress of restore operations

You can use the **LIST UTILITIES** command to monitor restore operations on a database.

Procedure

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

```
LIST UTILITIES SHOW DETAIL
```

Results

For restore operations, an initial estimate is not given. Instead, UNKNOWN is specified. As each buffer is read from the image, the actual number of bytes read is updated. For automatic incremental restore operations where multiple images might be restored, the progress is tracked by using phases. Each phase represents an image to be restored from the incremental chain. Initially, only one phase is indicated. After the first image is restored, the total number of phases will be indicated. As each image is restored the number of phases completed is updated, as is the number of bytes processed.

Example

The following is an example of the output for monitoring the performance of a restore operation:

```
ID                               = 6
Type                             = RESTORE
Database Name                    = SAMPLE
Partition Number                 = 0
Description                      = db
Start Time                       = 08/04/2011 12:24:47.494191
State                            = Executing
Invocation Type                  = User
Progress Monitoring:
      Completed Work             = 4096 bytes
      Start Time                 = 08/04/2011 12:24:47.494197
```

Optimizing restore performance

When you perform a restore operation, DB2 database products will automatically choose an optimal value for the number of buffers, the buffersize and the parallelism settings. The values will be based on the amount of utility heap memory available, the number of processors available and the database configuration.

Therefore, depending on the amount of storage available on your system, you should consider allocating more memory by increasing the **util_heap_sz** configuration parameter. The objective is to minimize the time it takes to complete a restore operation. Unless you explicitly enter a value for the following **RESTORE DATABASE** command parameters, DB2 database products will select one for them:

- **WITH** *num-buffers* **BUFFERS**
- **PARALLELISM** *n*
- **BUFFER** *buffer-size*

For restore operations, a multiple of the buffer size used by the backup operation will always be used. You can specify a buffer size when you issue the **RESTORE DATABASE** command but you need to make sure that it is a multiple of the backup buffer size.

You can also choose to do any of the following to reduce the amount of time required to complete a restore operation:

- Increase the restore buffer size.

  The restore buffer size must be a positive integer multiple of the backup buffer size specified during the backup operation. If an incorrect buffer size is specified, the buffers allocated will be the smallest acceptable size.
- Increase the number of buffers.

  The value you specify must be a multiple of the buffersize that was used for the backup, otherwise it will be rounded down to the closest multiple of the backup buffersize.
- Increase the value of the **PARALLELISM** parameter.

  This will increase the number of buffer manipulators (BM) that will be used to write to the database during the restore operation.
- Increase the utility heap size

  This will increase the memory that can be used simultaneously by the other utilities.

Chapter 48. ROLLFORWARD DATABASE command

Use the ROLLFORWARD DATABASE command to recover transactions that were logged after the last backup command. Database logging must be enabled for these commands to be effective.

The simplest form of the **ROLLFORWARD DATABASE** command requires only that you specify the alias name of the database that you want to rollforward recover, as in the following example:

```
db2 ROLLFORWARD DB sample
```

In IBM Data Studio Version 3.1 or later, you can use the task assistant for rolling forward databases. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

The following is one approach you can use to perform rollforward recovery:

1. Invoke the rollforward utility without the **STOP** option.
2. Invoke the rollforward utility with the **QUERY STATUS** option

   If you specify recovery to the end of the logs, the **QUERY STATUS** option can indicate that one or more log files are missing, if the returned point in time is earlier than you expect.

   If you specify point-in-time recovery, the **QUERY STATUS** option helps you to ensure that the rollforward operation completes at the correct point.
3. Invoke the rollforward utility with the **STOP** option. After the operation stops, it is not possible to roll additional changes forward.

An alternate approach you can use to perform rollforward recovery is the following:

1. Invoke the rollforward utility with the **AND STOP** option.
2. The need to take further steps depends on the outcome of the rollforward operation:
   - If it is successful, the rollforward is complete and the database is connectable and usable. At this point, it is not possible to roll additional changes forward.
   - If any errors were returned, take whatever action is required to fix the problem. For example, if there is a missing log file: find the log file, or if there are retrieve errors: ensure that log archiving is working. Then reissue the rollforward utility with the **AND STOP** option.

A database must be restored successfully (using the restore utility) before it can be rolled forward, but a table space does not. A table space can be temporarily put in rollforward pending state, but not require a restore operation to undo it (following a power interruption, for example).

When the rollforward utility is invoked:

- If the database is in rollforward pending state, the database is rolled forward. Any table spaces that were restored from backup images that were taken after the database backup image, and are currently in rollforward pending state are also rolled forward. Any table spaces that were taken prior to the database level backup and restored after the database level backup was restored remain in rollforward pending state. You must issue a subsequent table space level rollforward to recover them.

- If the database is *not* in rollforward pending state, but table spaces in the database *are* in rollforward pending state:
  - If you specify a list of table spaces, only those table spaces are rolled forward.
  - If you do not specify a list of table spaces, all table spaces that are in rollforward pending state are rolled forward.

A database rollforward operation runs offline. The database is not available for use until the rollforward operation completes successfully, and the operation cannot complete unless the **STOP** option was specified when the utility was invoked.

A table space rollforward operation can run offline. The database is not available for use until the rollforward operation completes successfully. This occurs if the end of the logs is reached, or if the **STOP** option was specified when the utility was invoked.

You can perform an *online* rollforward operation on table spaces, as long as SYSCATSPACE is not included. When you perform an online rollforward operation on a table space, the table space is not available for use, but the other table spaces in the database *are* available.

When you first create a database, it is enabled for circular logging only. This means that logs are reused, rather than being saved or archived. With circular logging, rollforward recovery is not possible: only crash recovery or version recovery can be done. Archived logs document changes to a database that occur after a backup was taken. You enable log archiving (and rollforward recovery) by setting the **logarchmeth1** database configuration parameter to a value other than its default of `OFF`. When you set **logarchmeth1** to a value other than `OFF`, the database is placed in backup pending state, and you must take an offline backup of the database before it can be used again.

Note: Entries are made in the recovery history file for each log file that is used in a rollforward operation.
In this example, the command returns:

In a partitioned database environment and a DB2 pureScale environment, this status information is returned for each database partition or member:

```
db2 rollforward db mydb to end of logs

                                 Rollforward Status

 Input database alias                  = mydb
 Number of members have returned status = 3

 Member ID Rollforward   Next log      Log files processed        Last committed transaction
           status        to be read
 --------- ------------  ------------  -------------------------  --------------------------
         0  DB  working  S0000001.LOG  S0000000.LOG-S0000000.LOG  2009-05-06-15.28.11.000000 UTC
         1  DB  working  S0000010.LOG  S0000000.LOG-S0000009.LOG  2009-05-06-15.28.20.000000 UTC
         2  DB  working  S0000005.LOG  S0000000.LOG-S0000004.LOG  2009-05-06-15.27.33.000000 UTC

DB20000I  The ROLLFORWARD command completed successfully.
```

Authorization required for rollforward

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the rollforward utility.

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects.

Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

Using rollforward

Use the **ROLLFORWARD DATABASE** command to apply transactions that were recorded in the database log files to a restored database backup image or table space backup image.

Before you begin

You should not be connected to the database that is to be rollforward recovered. The rollforward utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the rollforward operation.

About this task

Do not restore table spaces without canceling a rollforward operation that is in progress. Otherwise, you might have a table space set in which some table spaces are in rollforward in progress state, and some table spaces are in rollforward pending state. A rollforward operation that is in progress only operates on the tables spaces that are in rollforward in progress state.

The database can be local or remote.

The following restrictions apply to the rollforward utility:

- You can invoke only one rollforward operation at a time. If there are many table spaces to recover, you can specify all of them in the same operation.
- If you have renamed a table space following the most recent backup operation, ensure that you use the new name when rolling the table space forward. The previous table space name is not recognized.
- You cannot cancel a rollforward operation that is running. You can only cancel a rollforward operation that has completed, but for which the **STOP** parameter has not been specified, or a rollforward operation that has failed before completing.
- You cannot *continue* a table space rollforward operation to a point in time, specifying a time stamp that is less than the previous one. If a point in time is not specified, the previous one is used. You can issue a rollforward operation that ends at a specified point in time by just specifying STOP, but this is only allowed if the table spaces involved were all restored from the same offline backup image. In this case, no log processing is required. If you start another rollforward operation with a different table space list before the in-progress rollforward operation is either completed or cancelled, an error message (SQL4908) is returned. Invoke the **LIST TABLESPACES** command on all database partitions (or use the MON_GET_TABLESPACE table function) to determine which table spaces are currently being rolled forward (rollforward in progress state), and which table spaces are ready to be rolled forward (rollforward pending state). You have three options:
  - Finish the in-progress rollforward operation on all table spaces.

  - Finish the in-progress rollforward operation on a subset of table spaces. (This might not be possible if the rollforward operation is to continue to a specific point in time, which requires the participation of all database partitions.)
  - Cancel the in-progress rollforward operation.
- In a partitioned database environment, the rollforward utility must be invoked from the catalog partition of the database.
- Point in time rollforward of a table space was introduced in DB2 Version 9.1 clients. You should upgrade to Version 10.1 any clients in order to roll a table space forward to a point in time.
- You cannot roll forward logs from a previous release version.

Procedure

To invoke the rollforward utility, use the:

- **ROLLFORWARD DATABASE** command, or
- db2Rollforward application programming interface (API).
- Open the task assistant inIBM Data Studio for the **ROLLFORWARD DATABASE** command.

Example

The following is an example of the **ROLLFORWARD DATABASE** command issued through the CLP:

```
db2 rollforward db sample to end of logs and stop
```

Rollforward sessions - CLP examples

You can issue rollforward commands from the Command Line Prompt. Before issuing a rollforward command, you might find it helpful to review some sample sessions.

Example 1

The ROLLFORWARD DATABASE command permits specification of multiple operations at once, each being separated with the keyword AND. For example, to roll forward to the end of logs, and complete, the separate commands are:

```
   db2 rollforward db sample to end of logs
   db2 rollforward db sample complete
```

can be combined as follows:

```
   db2 rollforward db sample to end of logs and complete
```

Although the two are equivalent, it is recommended that such operations be done in two steps. It is important to verify that the rollforward operation has progressed as expected before you stop it, so that you do not miss any logs.

If the rollforward command encounters an error, the rollforward operation will not complete. The error will be returned, and you will then be able to fix the error and reissue the command. If, however, you are unable to fix the error, you can force the rollforward to complete by issuing the following:

```
   db2 rollforward db sample complete
```

This command brings the database online at the point in the logs before the failure.

Example 2

Roll the database forward to the end of the logs (two table spaces have been restored):

```
db2 rollforward db sample to end of logs
db2 rollforward db sample to end of logs and stop
```

These two statements are equivalent. Neither AND STOP or AND COMPLETE is needed for table space rollforward recovery to the end of the logs. Table space names are not required. If not specified, all table spaces requiring rollforward recovery will be included. If only a subset of these table spaces is to be rolled forward, their names must be specified.

Example 3

After three table spaces have been restored, roll one forward to the end of the logs, and the other two to a point in time, both to be done online:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online
db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   tablespace(TBS2, TBS3) online
```

Note that two rollforward operations cannot be run concurrently. The second command can only be invoked after the first rollforward operation completes successfully.

Example 4

After restoring the database, roll forward to a point in time, using OVERFLOW LOG PATH to specify the directory where the user exit saves archived logs:

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   overflow log path (/logs)
```

Example 5

In the following example, there is a database called sample. The database is backed up and the recovery logs are included in the backup image; the database is restored; and the database is rolled forward to the end of backup timestamp.

Back up the database, including the recovery logs in the backup image:

```
db2 backup db sample online include logs
```

Restore the database using that backup image:

```
db2 restore db sample
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup
```

Example 6 (partitioned database environments)

There are three database partitions: 0, 1, and 2. Table space TBS1 is defined on all database partitions, and table space TBS2 is defined on database partitions 0 and 2. After restoring the database on database partition 1, and TBS1 on database partitions 0 and 2, roll the database forward on database partition 1:

```
db2 rollforward db sample to end of logs and stop
```

This returns warning SQL1271 (“Database is recovered but one or more table spaces are offline on database partitions 0 and 2.”).

```
db2 rollforward db sample to end of logs
```

This rolls TBS1 forward on database partitions 0 and 2. The clause TABLESPACE(TBS1) is optional in this case.

Example 7 (partitioned database environments)

In the following example, there is a partitioned database called sample. All the database partitions are backed up with a single system view backup; the database is restored on all database partitions; and the database is rolled forward to the end of backup timestamp.

Perform a single system view (SSV) backup:

```
db2 backup db sample on all nodes online include logs
```

Restore the database on all database partitions:

```
db2_all "db2 restore db sample taken at 1998-04-03-14.21.56"
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup on all nodes
```

Example 8 (partitioned database environments)

In the following example, there is a partitioned database called sample. All the database partitions are backed up with one command using db2_all; the database is restored on all database partitions; and the database is rolled forward to the end of backup timestamp.

Back up all the database partitions with one command using db2_all:

```
db2_all "db2 backup db sample include logs to //dir/"
```

Restore the database on all database partitions:

```
db2_all "db2 restore db sample from //dir/"
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup on all nodes
```

Example 9 (partitioned database environments)

After restoring table space TBS1 on database partitions 0 and 2 only, roll TBS1 forward on database partitions 0 and 2:

```
db2 rollforward db sample to end of logs
```

Database partition 1 is ignored.

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1. Reports SQL4906N.

```
db2 rollforward db sample to end of logs on
   dbpartitionnums (0, 2) tablespace(TBS1)
```

This completes successfully.

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1; all pieces must be rolled forward together.

Note: With table space rollforward to a point in time, the dbpartitionnum clause is not accepted. The rollforward operation must take place on all the database partitions on which the table space resides.

After restoring TBS1 on database partition 1:

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   tablespace(TBS1)
```

This completes successfully.

Example 10 (partitioned database environments)

After restoring a table space on all database partitions, roll forward to PIT2, but do not specify `AND STOP`. The rollforward operation is still in progress. Cancel and roll forward to PIT1:

```
   db2 rollforward db sample to pit2 tablespace(TBS1)
   db2 rollforward db sample cancel tablespace(TBS1)

 ** restore TBS1 on all dbpartitionnums **

   db2 rollforward db sample to pit1 tablespace(TBS1)
   db2 rollforward db sample stop tablespace(TBS1)
```

Example 11 (partitioned database environments)

Rollforward recover a table space that resides on eight database partitions (3 to 10) listed in the `db2nodes.cfg` file:

```
db2 rollforward database dwtest to end of logs tablespace (tssprodt)
```

This operation to the end of logs (not point in time) completes successfully. The database partitions on which the table space resides do not have to be specified. The utility defaults to the `db2nodes.cfg` file.

Example 12 (partitioned database environments)

Rollforward recover six small table spaces that reside on a single database partition database partition group (on database partition 6):

```
db2 rollforward database dwtest to end of logs on dbpartitionnum (6)
   tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

This operation to the end of logs (not point in time) completes successfully.

Example 13 (Partitioned tables - Rollforward to end of log on all data partitions)

A partitioned table is created using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, and attaches data partitions from the table in tbsp5. All table spaces can be rolled forward to END OF LOGS.

```
db2 rollforward db PBARDB to END OF LOGS and stop
   tablespace(tbsp0, tbsp1, tbsp2, tbsp3, tbsp4, tbsp5)
```

This completes successfully.

Example 14 (Partitioned tables - Rollforward to end of logs on one table space)

A partitioned table is created initially using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, and attaches data partitions from the table in tbsp5. Table space tbsp4 becomes corrupt and requires a restore and rollforward to end of logs.

```
db2 rollforward db PBARDB to END OF LOGS and stop tablespace(tbsp4)
```

This completes successfully.

Example 15 (Partitioned tables - Rollforward to PIT of all data partitions including those added, attached, detached or with indexes)

A partitioned table is created using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, attaches data partitions from the table in tbsp5, and detaches data partitions from tbsp1. The user performs a rollforward to PIT with all the table spaces used by the partitioned table including those table spaces specified in the INDEX IN clause.

```
db2 rollforward db PBARDB to 2005-08-05-05.58.53 and stop
   tablespace(tbsp0, tbsp1, tbsp2, tbsp3, tbsp4, tbsp5)
```

This completes successfully.

Example 16 (Partitioned tables - Rollforward to PIT on a subset of the table spaces)

A partitioned table is created using three table spaces (tbsp1, tbsp2, tbsp3). Later, the user detaches all data partitions from tbsp3. The rollforward to PIT is only permitted on tbsp1 and tbsp2.

```
db2 rollforward db PBARDB to 2005-08-05-06.02.42 and stop
   tablespace( tbsp1, tbsp2)
```

This completes successfully.

Rolling forward changes in a table space

If the database is enabled for rollforward recovery, you have the option of backing up, restoring, and rolling forward table spaces instead of the entire database.

You can roll forward changes to a table space independently of other table spaces in your database, or you can roll forward changes to all table spaces at the same time.

Implementing a recovery strategy for individual table spaces can save time because it takes less time to recover a portion of the database than it does to recover the entire database. For example, if a disk is bad, and it contains only one table space, you can restore that table space and roll it forward without having to recover the entire database, and without impacting user access to the rest of the database, unless the damaged table space contains the system catalog tables; in this situation, you cannot connect to the database. (You can restore the system catalog table space independently if a table space-level backup image containing the system catalog table space is available.) Table space-level backups also allow you to back up critical parts of the database more frequently than other parts, and requires less time than backing up the entire database.

After a table space is restored, it is always in rollforward pending state. To make the table space usable, you must perform rollforward recovery on it. In most cases, you have the option of rolling forward to the end of the logs, or rolling forward to a point in time. You cannot, however, roll table spaces containing system catalog tables forward to a point in time. These table spaces must be rolled forward to the end of the logs to ensure that all table spaces in the database remain consistent.

Ensure that the **DB2_COLLECT_TS_REC_INFO** registry variable is set to `ON` (the default) if you want to skip the log files known not to contain any log records affecting the table space. This registry variable must be set before the log files are created and used so that the information required for skipping log files is collected. If **DB2_COLLECT_TS_REC_INFO** is set to `OFF`, all log files are processed even if they do not contain log records that affect that table space when that table space is rolled forward.

The table space change history file (`DB2TSCHG.HIS`), which is located in the database directory, tracks which logs to process for each table space. You can view the contents of this file with the **db2logsForRfwd** utility, and delete entries from it with the **PRUNE HISTORY** command. During a database restore operation, the `DB2TSCHG.HIS` file is restored from the backup image and then brought up to date during the database rollforward operation. If no information is available for a log file, it is treated as though it is required for the recovery of every table space.

Because information for each log file is flushed to disk after the log becomes inactive, this information can be lost as a result of a crash. To prevent this loss from occurring, if a recovery operation begins in the middle of a log file, the entire log is treated as though it contains modifications to every table space in the system. All active logs are processed and the information for them is rebuilt. If information for older or archived log files is lost in a crash situation and no information for them exists in the data file, they are treated as though they contain modifications for every table space during the table space recovery operation.

Before you roll a table space forward, use the MON_GET_TABLESPACE table function to determine the *minimum recovery time*, which is the earliest point in time to which the table space can be rolled forward. The minimum recovery time is updated when data definition language (DDL) statements are run against the table space, or against tables in the table space. The table space must be rolled forward to at least the minimum recovery time so that it becomes synchronized with the information in the system catalog tables. If you are recovering more than one table space, the table spaces must be rolled forward to at least the highest minimum recovery time of all the table spaces that are being recovered. You cannot roll forward a table space to a time that is earlier than the backup timestamp. In a partitioned database environment, you must roll forward the table spaces to at least the highest minimum recovery time of all the table spaces on all database partitions.

If you are rolling table spaces forward to a point in time, and a table is contained in multiple table spaces, all of these table spaces must be rolled forward simultaneously. If, for example, the table data is contained in one table space, and the index for the table is contained in another table space, you must roll both table spaces forward simultaneously to the same point in time.

If the data and the long objects in a table are in separate table spaces, and the long object data was reorganized, the table spaces for both the data and the long objects must be restored and rolled forward together. Take a backup of the affected table spaces after the table is reorganized.

If you want to roll forward a table space to a point in time, and a table in the table space is either:

- an underlying table for a materialized query or staging table that is in another table space
- a materialized query or staging table for a table in another table space

then roll both table spaces forward to the same point in time. If you do not, the materialized query or staging table is placed in set integrity pending state at the end of the rollforward operation. The materialized query table needs to be fully refreshed, and the staging table is marked as incomplete.

If you want to roll forward a table space to a point in time, and a table in the table space participates in a referential integrity relationship with another table that is contained in another table space, roll forward both table spaces simultaneously to the same point in time. If you do not roll forward both table spaces, the child table in the referential integrity relationship is placed in set integrity pending state at the end of the rollforward operation. When the child table is later checked for constraint violations, a check on the entire table is required. If any of the following tables exist, they are also placed in set integrity pending state with the child table:

- any descendant materialized query tables for the child table
- any descendant staging tables for the child table
- any descendant foreign key tables of the child table

These tables require full integrity processing to bring them out of the set integrity pending state. If you roll forward both table spaces simultaneously, the constraint remains active at the end of the point-in-time rollforward operation.

Ensure that a point-in-time table space rollforward operation does not cause a transaction to be rolled back in some table spaces, and committed in others. This inconsistency can happen in the following cases:

- A point-in-time rollforward operation is performed on a subset of the table spaces that were updated by a transaction, and that point in time precedes the time at which the transaction was committed.
- Any table that is contained in the table space being rolled forward to a point in time has an associated trigger, or is updated by a trigger that affects table spaces other than the one that is being rolled forward.

The solution is to find a suitable point in time that prevents this from happening.

You can issue the **QUIESCE TABLESPACES FOR TABLE** command to create a transaction-consistent point in time for rolling table spaces forward. The quiesce request (in share, intent to update, or exclusive mode) waits (through locking) for all running transactions against those table spaces to complete, and blocks new requests. When the quiesce request is granted, the table spaces are in a consistent state. To determine a suitable time to stop the rollforward operation, you can look in the recovery history file to find quiesce points, and check whether they occur after the minimum recovery time.

After a table space point-in-time rollforward operation completes, the table space is put in backup pending state. You must take a backup of the table space because all updates made to it between the point in time to which you rolled forward and the current time were removed. You can no longer roll forward the table space to the current time from a previous database- or table space-level backup image. The following example shows why the table space-level backup image is required, and how it is used. (To make the table space available, you can either back up the entire database, the table space that is in backup pending state, or a set of table spaces that includes the table space that is in backup pending state.)

```
Database                            Time of rollforward of    Restore
backup                              table space TABSP1 to     database.
                                    T2. Back up TABSP1.       Roll forward
                                                              to end of logs.
T1             T2                   T3                        T4
|              |                    |                         |
|              |                    |                         |
|----------------------------------------------------------------------------
               | Logs are not
                 applied to TABSP1
                 between T2 and T3
                 when it is rolled
                 forward to T2.
```

Figure 66. Table space backup requirement

In the preceding example, the database is backed up at time T1. Then, at time T3, table space TABSP1 is rolled forward to a specific point in time (T2), The table space is backed up after time T3. Because the table space is in backup pending state, this backup operation is mandatory. The timestamp of the table space backup image is after time T3, but the table space is at time T2. Log records from between T2 and T3 are not applied to TABSP1. At time T4, the database is restored, using the backup image that was created at T1, and rolled forward to the end of the logs. Table space TABSP1 is put in restore pending state at time T3, because the database manager assumes that operations were performed on TABSP1 between T3 and T4 without the log changes between T2 and T3 being applied to the table space. If these log changes were in fact applied as part of the rollforward operation against the database, this assumption would be incorrect. The table space-level backup that must be taken after the table space is rolled forward to a point in time allows you to roll forward that table space past a previous point-in-time rollforward operation (T3 in the example).

Assuming that you want to recover table space TABSP1 to T4, you would restore the table space from a backup image that was taken after T3 (either the required backup, or a later one), then roll forward TABSP1 to the end of the logs.

In the preceding example, the most efficient way of restoring the database to time T4 would be to perform the required steps in the following order:

1. Restore the database.
2. Restore the table space.
3. Roll forward the database.

Because you restore the table space before you roll forward the database, resources are not used to apply log records to the table space when the database is rolled forward.

If you cannot find the TABSP1 backup image that follows time T3, or you want to restore TABSP1 to T3 (or earlier), you can do one of the following actions:

- Roll forward the table space to T3. You do not need to restore the table space again because it was restored from the database backup image.

- Restore the table space again by restoring the database backup that was taken at time T1, and then roll forward the table space to a time that precedes time T3.
- Drop the table space.

In a partitioned database environment:

- You must simultaneously roll forward all parts of a table space to the same point in time at the same time. This ensures that the table space is consistent across database partitions.
- If some database partitions are in rollforward pending state, and on other database partitions, some table spaces are in rollforward pending state (but the database partitions are not), you must first roll forward the database partitions, and then roll forward the table spaces.
- If you intend to roll forward a table space to the end of the logs, you do not have to restore it at each database partition; you must restore it at the database partitions that require recovery. If you intend to roll forward a table space to a point in time, however, you must restore it at each database partition.

In a database with partitioned tables:

- If you are rolling a table space that contains any piece of a partitioned table forward to a point in time, you must also roll forward all of the other table spaces in which that table resides to the same point in time. However, rolling forward a single table space containing a piece of a partitioned table to the end of logs is allowed. If a partitioned table has any attached, detached, or dropped data partitions, then a point-in-time rollforward operation must also include all table spaces for these data partitions. In order to determine if a partitioned table has any attached, detached, or dropped data partitions, query the SYSCAT.DATAPARTITIONS catalog view.

Database rollforward operations in a DB2 pureScale environment

In a DB2 pureScale environment, each member has its own log stream; however, log streams from all members are required for successful execution of the **ROLLFORWARD DATABASE** command.

During a database rollforward operation, log records from all of the log streams are merged and replayed to make the database consistent. The point in time that you specify on the **ROLLFORWARD DATABASE** command is relative to the merged log stream. To restore the database to a consistent state, the specified time must be later than the *minimum recovery time* (MRT). The MRT is the earliest time during a rollforward operation when objects that are listed in the database catalog match the objects that physically exist on disk. For example, if you are restoring from an image that was created during an online backup operation, the specified point in time for the rollforward operation must be later than the time at which the online backup operation completed. This will ensure database consistency.

The specified point in time for the subsequent database rollforward operation must be greater than or equal to the MRT in the merged log stream; otherwise, the rollforward operation fails (SQL1276N), and the timestamp of the MRT is returned with the error message. Alternatively, you can use the END OF BACKUP option to automatically roll forward to the MRT.

It is recommended that the member clocks be synchronized; however, it might not be possible to synchronize them at all times. This can result in log records having the same time stamp, and merged log streams with log records that appear to be out of time stamp order. In a DB2 pureScale environment, a point-in-time database rollforward operation stops when it encounters the first log record whose time stamp is greater than the specified time stamp from any log stream, and it has processed the log record that corresponds to the MRT for the database.

An incomplete or interrupted rollforward operation leaves the database in rollforward pending state. In this case, issue another **ROLLFORWARD DATABASE** command. In a DB2 pureScale environment, subsequent **ROLLFORWARD DATABASE** commands can be run on the same or on a different member.

In a DB2 pureScale environment, if you want to perform a database restore operation into a new database using an online database backup image, the correct approach depends on whether all of the log files are available, or only log files from the backup image are available.

- If pre-existing log files or archived log files can be accessed, the following rollforward operation is appropriate:

  ```
  db2 rollforward db dbname to end of logs and stop
  ```

  Note: Before taking a backup, you need to ensure that the log archiving path is set to a shared directory so that all the members are able to access the logs for subsequent rollforward operations. If the archive path is not accessible from the member on which the rollforward is being executed, SQL1273N is returned. The following command is an example of how to set the log path to the shared directory:

  ```
  db2 update db cfg using logarchmeth1
        DISK:/db2fs/gpfs1/svtdbm5/svtdbm5/ArchiveLOGS
  ```

  (where *gpfs1* is the shared directory for the members and *ArchiveLOGS* is the actual directory that archives the logs.

- If the only log files that can be accessed come from the backup image, the following rollforward operation is appropriate:

  ```
  db2 rollforward db dbname to end of backup and stop
  ```

  This command replays all required log records to achieve the consistent database state that was in effect when the backup operation ended. You can also use this command if pre-existing log files or archived log files can be accessed, but it will stop at the point at which the backup operation ended; it will not use any extra logs that were generated after the backup operation ended.

  A **ROLLFORWARD DATABASE** command specifying the END OF LOGS option in this case would return SQL1273N. A subsequent **ROLLFORWARD DATABASE** command with the STOP option is successful, and the database will be available, if the missing log files are not needed. However, if the missing log files are needed (and it is not safe to stop), the rollforward operation will again return SQL1273N.

Example

Suppose that there are two members, M1 and M2. M2's clock is ahead of M1's clock by five seconds. M2's log stream contains the following log records:

A1 at 2010-04-03-14.21.56

A2 at 2010-04-03-14.21.56

B at 2010-04-03-14.21.58

C at 2010-04-03-14.22.01

M1's log stream contains the following log records:

D at 2010-04-03-14.21.55

E at 2010-04-03-14.21.56

F at 2010-04-03-14.21.57

The minimum recovery time (MRT) for the database on M2 is at time 2010-04-03-14.21.55. Because M1's clock is five seconds slow, log records D, E, and F appear later in the merged log stream:

```
MRT: 2010-04-03-14.21.55 (M2)
A1:  2010-04-03-14.21.56 (M2)
A2:  2010-04-03-14.21.56 (M2)
B:   2010-04-03-14.21.58 (M2)
D:   2010-04-03-14.21.55 (M1) --> corresponding time on M2 is 14.22.00
C:   2010-04-03-14.22.01 (M2)
E:   2010-04-03-14.21.56 (M1) --> corresponding time on M2 is 14.22.01
F:   2010-04-03-14.21.57 (M1) --> corresponding time on M2 is 14.22.02
```

The alphabetic characters (A1, A2, B, and so on) represent the order in which the corresponding log records were actually written at run time (across members). Note that log records A1 and A2 from member M2 have the same time stamp; this can happen when the DB2 data server tries to optimize performance by including the commit log record from multiple transactions when data is written from the log buffer to a log file.

The following command returns SQL1276N (Database "test" cannot be brought out of rollforward pending state until rollforward has passed a point in time greater than or equal to "2010-04-03-14.21.55"):

```
db2 rollforward db test to 2010-04-03-14.21.54
```

But the following command rolls forward the database up to and including log record A2:

```
db2 rollforward db test to 2010-04-03-14.21.56
```

Because log records A1 and A2 both have a time stamp that is less than or equal to the time that was specified in the command, both are replayed. Log record B, whose time stamp (2010-04-03-14.21.58) is greater than the specified value (2010-04-03-14.21.56), stops the rollforward operation and is not replayed. Log record D is not replayed either, even though its time stamp is less than the specified value, because log record B's higher value (2010-04-03-14.21.58) was encountered first. The following command rolls forward the database up to and including log record D:

```
db2 rollforward db test to 2010-04-03-14.21.58
```

Log record C, whose time stamp (2010-04-03-14.22.01) is greater than the specified value (2010-04-03-14.21.58), stops the rollforward operation and is not replayed. Log record E is not replayed either, even though its time stamp is less than the specified value.

Monitoring a rollforward operation

You can use the **db2pd** or the **LIST UTILITIES** command to monitor the progress of rollforward operations on a database.

Procedure

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

  ```
  LIST UTILITIES SHOW DETAIL
  ```

- Issue the **db2pd** command and specify the **-recovery** parameter:

```
db2pd -recovery
```

Results

For rollforward recovery, there are two phases of progress monitoring: FORWARD and BACKWARD. During the FORWARD phase, log files are read and the log records are applied to the database. For rollforward recovery, when this phase begins UNKNOWN is specified for the total work estimate. The amount of work processed in bytes is updated as the process continues.

During the BACKWARD phase, any uncommitted changes applied during the FORWARD phase are rolled back. An estimate for the amount of log data to be processed, in bytes, is provided. The amount of work processed, in bytes, is updated as the process continues.

Example

The following is an example of the output for monitoring the performance of a rollforward operation using the **db2pd** command:

```
Recovery:
Recovery Status     0x00000401
Current Log         S0000005.LOG
Current LSN         0000001F07BC
Current LSO         000002551BEA
Job Type            ROLLFORWARD RECOVERY
Job ID              7
Job Start Time      (1107380474) Wed Feb  2 16:41:14 2005
Job Description     Database Rollforward Recovery
Invoker Type        User
Total Phases        2
Current Phase       1

Progress:
Address            PhaseNum Description StartTime                CompletedWork  TotalWork
0x0000000200667160 1        Forward     Wed Feb  2 16:41:14 2005 2268098 bytes  Unknown
0x0000000200667258 2        Backward    NotStarted               0 bytes        Unknown
```

The following is an example of the output for monitoring the performance of a database rollforward operation using the **LIST UTILITIES** command with the SHOW DETAIL option:

```
ID                               = 7
Type                             = ROLLFORWARD RECOVERY
Database Name                    = TESTDB
Member Number                    = 0
Description                      = Database Rollforward Recovery
Start Time                       = 01/11/2012 16:56:53.770404
State                            = Executing
Invocation Type                  = User
Progress Monitoring:
   Estimated Percentage Complete = 50
   Phase Number                  = 1
      Description                = Forward
      Total Work                 = 928236 bytes
      Completed Work             = 928236 bytes
      Start Time                 = 01/11/2012 16:56:53.770492

   Phase Number [Current]        = 2
      Description                = Backward
      Total Work                 = 928236 bytes
      Completed Work             = 0 bytes
      Start Time                 = 01/11/2012 16:56:56.886036
```

The following is an example of the output for monitoring the performance of a table space rollforward operation using the **LIST UTILITIES** command with the SHOW DETAIL option:

```
ID                               = 17
Type                             = ROLLFORWARD RECOVERY
Database Name                    = TESTDB
Member Number                    = 0
Description                      = Offline Tablespace Rollforward Recovery: 3
Start Time                       = 01/11/2012 17:04:27.269171
State                            = Executing
Invocation Type                  = User
Progress Monitoring:
   Estimated Percentage Complete = 63
   Phase Number                  = 1
      Description                = Forward
      Total Work                 = 142
      Completed Work             = 90
      Start Time                 = 01/11/2012 17:04:27.269283

   Phase Number [Current]        = 2
      Description                = Backward
      Total Work                 = 0
      Completed Work             = 0
      Start Time                 = Not Started
```

Chapter 49. High availability disaster recovery (HADR)

The high availability disaster recovery (HADR) feature provides a high availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the *primary database*, to one or more target databases, called the *standby databases*.

A partial site failure can be caused by a hardware, network, or software (DB2 database system or operating system) failure. Without HADR, a partial site failure requires restarting the database management system (DBMS) server that contains the database. The length of time that it takes to restart the database and the server where it is located is unpredictable. It can take several minutes before the database is brought back to a consistent state and made available. With HADR, a standby database can take over in seconds. Further, you can redirect the clients that used the original primary database to the new primary database by using automatic client reroute or retry logic in the application.

A complete site failure can occur when a disaster, such as a fire, causes the entire site to be destroyed. However, because HADR uses TCP/IP for communication between the primary and standby databases, they can be situated in different locations. For example, the primary database might be located at your head office in one city, and a standby database might be located at your sales office in another city. If a disaster occurs at the primary site, data availability is maintained by having the remote standby database take over as the primary database with full DB2 functionality. After a takeover operation occurs, you can bring the original primary database back up and return it to its primary database status; this is known as *failback*. You can initiate a failback if you can make the old primary database consistent with the new primary database. After you reintegrate the old primary database into the HADR setup as a standby database, you can switch the roles of the databases to enable the original primary database to once again be the primary database.

With HADR, you base the level of protection from potential loss of data on your configuration and topology choices. Some of the key choices that you must make are as follows:

What level of synchronization will you use?

Standby databases are synchronized with the primary database through log data that is generated on the primary and shipped to the standbys. The standbys constantly roll forward through the logs. You can choose from four different synchronization modes. In order of most to least protection, these are SYNC, NEARSYNC, ASYNC, and SUPERASYNC. For more information, see “High Availability Disaster Recovery (HADR) synchronization mode” on page 849.

Will you use a peer window?

The peer window feature specifies that the primary and standby databases are to behave as though they are still in peer state for a configured amount of time if the primary loses the HADR connection in peer state. If primary fails in peer or this "disconnected peer" state, the failover to standby will have zero data loss. This feature provides the greatest protection. For more information, see “Setting the hadr_timeout and hadr_peer_window database configuration parameters” on page 911.

How many standbys will you deploy?

With HADR, you can use either single standby mode or multiple standby mode. With multiple standbys, you can achieve both your high availability and disaster recovery objectives with a single technology. For more information, see "HADR multiple standby databases" on page 853.

There are a number of ways that you can use your HADR standby or standbys beyond their HA or DR purpose:

Reads on standby

You can use the reads on standby feature to direct read-only workload to one or more standby databases without affecting the HA or DR responsibility of the standby. This feature can help reduce the workload on the primary without affecting the main responsibility of the standby. For more information on this topic, see "HADR reads on standby feature" on page 876.

Unless you have reads on standby enabled, applications can access the current primary database only. If you have reads on standby enabled, read-only applications can be redirected to the standby. Applications connecting to the standby database do not affect the availability of the standby in the case of a failover.

Delayed replay

You can use delayed replay to specify that a standby database is to remain at an earlier point in time than the primary, in terms of log replay. If data is lost or corrupted on the primary, you can recovery this data on the time delayed standby. For more information, see "HADR delayed replay" on page 881.

Rolling updates and upgrades

Using an HADR setup, you can make various types of upgrades and DB2 fix pack updates to your databases without an outage. If you are using multiple standby mode enabled, you can perform an upgrade while at the same time keeping the protection provided by HADR. For more information, see "Performing rolling updates in a DB2 High Availability Disaster Recovery (HADR) environment" on page 884.

HADR might be your best option if most or all data in your database requires protection or if you perform DDL operations that must be automatically replicated on a standby database. However, HADR is only one of several replication solutions that are offered in the DB2 product family. The InfoSphere Federation Server software and the DB2 database system include SQL replication and Q replication solutions that you can also use, in some configurations, to provide high availability. These solutions maintain logically consistent copies of database tables at multiple locations. In addition, they provide flexibility and complex functionality such as support for column and row filtering, data transformation, and updates to any copy of a table. You can also use these solutions in partitioned database environments.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for setting up HADR. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

High Availability Disaster Recovery (HADR) synchronization mode

The HADR synchronization mode determines the degree of protection your DB2 High Availability Disaster Recovery (HADR) database solution has against transaction loss. The synchronization mode determines when the primary database server considers a transaction complete, based on the state of the logging on the standby database.

The more strict the synchronization mode configuration parameter value, the more protection your database solution has against transaction data loss, but the slower your transaction processing performance. You must balance the need for protection against transaction loss with the need for performance.

Figure 67 shows the DB2 HADR synchronization modes that are available and also when transactions are considered committed based on the synchronization mode chosen:

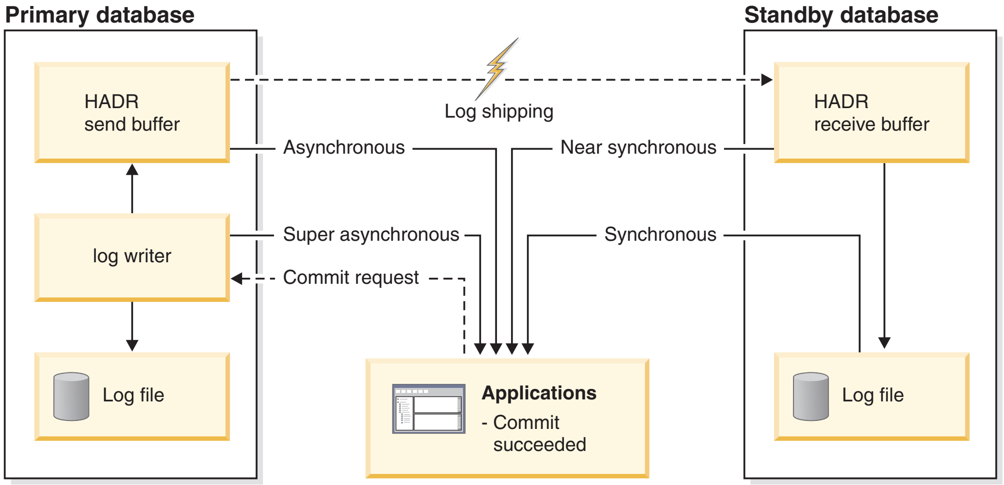


Figure 67. Synchronization modes for high availability and disaster recovery (HADR)

In multiple standby mode, the setting for **hadr_syncmode** does not need to be the same on the primary and standby databases. Whatever setting for **hadr_syncmode** is specified on a standby is considered its *configured synchronization mode*; this setting only has relevance if the standby becomes a primary. Instead, the standby is assigned an *effective synchronization mode*. For any auxiliary standby, the effective synchronization mode is always SUPERASYNC. For the principal standby, the effective synchronization mode is the primary's setting for **hadr_syncmode**. A standby's effective synchronization mode is the value that is displayed by any monitoring interface.

Use the **hadr_syncmode** database configuration parameter to set the synchronization mode. The following values are valid:

SYNC (synchronous)

This mode provides the greatest protection against transaction loss, and using it results in the longest transaction response time among the four modes.

In this mode, log writes are considered successful only when logs have been written to log files on the primary database and when the primary database has received acknowledgement from the standby database that the logs have also been written to log files on the standby database. The log data is guaranteed to be stored at both sites.

If the standby database crashes before it can replay the log records, the next time it starts it can retrieve and replay them from its local log files. If the primary database fails, a failover to the standby database guarantees that any transaction that has been committed on the primary database has also been committed on the standby database. After the failover operation, when the client reconnects to the new primary database, there can be transactions committed on the new primary database that were never reported as committed to the application on the original primary. This occurs when the primary database fails before it processes an acknowledgement message from the standby database. Client applications should consider querying the database to determine whether any such transactions exist.

If the primary database loses its connection to the standby database, what happens next depends on the configuration of the **hadr_peer_window** database configuration parameter. If **hadr_peer_window** is set to a non-zero time value, then upon losing connection with the standby database the primary database will move into disconnected peer state and continue to wait for acknowledgement from the standby database before committing transactions. If the **hadr_peer_window** database configuration parameter is set to zero, the primary and standby databases are no longer considered to be in peer state and transactions will not be held back waiting for acknowledgement from the standby database. If the failover operation is performed when the databases are not in peer or disconnected peer state, there is no guarantee that all of the transactions committed on the primary database will appear on the standby database.

If the primary database fails when the databases are in peer or disconnected peer state, it can rejoin the HADR pair as a standby database after a failover operation. Because a transaction is not considered to be committed until the primary database receives acknowledgement from the standby database that the logs have also been written to log files on the standby database, the log sequence on the primary will be the same as the log sequence on the standby database. The original primary database (now a standby database) just needs to catch up by replaying the new log records generated on the new primary database since the failover operation.

If the primary database is not in peer state when it fails, its log sequence might be different from the log sequence on the standby database. If a failover operation has to be performed, the log sequence on the primary and standby databases might be different because the standby database starts its own log sequence after the failover. Because some operations cannot be undone (for example, dropping a table), it is not possible to revert the primary database to the point in time when the new log sequence was created. If the log sequences are different and you issue the **START HADR** command with the **AS STANDBY** parameter on the original primary, you will receive a message that the command was successful. However, this message is issued before reintegration is attempted. If reintegration fails, pair validation messages will be issued to the administration log and the diagnostics log on both the primary and the standby. The reintegrated standby will remain the standby, but the primary will reject the standby during pair validation causing the standby database to shut down. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the

TAKEOVER HADR command without specifying the **BY FORCE** parameter. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

NEARSYNC (near synchronous)

While this mode has a shorter transaction response time than synchronous mode, it also provides slightly less protection against transaction loss.

In this mode, log writes are considered successful only when the log records have been written to the log files on the primary database and when the primary database has received acknowledgement from the standby system that the logs have also been written to main memory on the standby system. Loss of data occurs only if both sites fail simultaneously and if the target site has not transferred to nonvolatile storage all of the log data that it has received.

If the standby database crashes before it can copy the log records from memory to disk, the log records will be lost on the standby database. Usually, the standby database can get the missing log records from the primary database when the standby database restarts. However, if a failure on the primary database or the network makes retrieval impossible and a failover is required, the log records will never appear on the standby database, and transactions associated with these log records will never appear on the standby database.

If transactions are lost, the new primary database is not identical to the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database cannot to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records (because both the primary and standby databases have failed), the log sequences on the primary and standby databases will be different and attempts to restart the original primary database as a standby database without first performing a restore operation will fail. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the **TAKEOVER HADR** command without specifying the **BY FORCE** parameter. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

ASYNC (asynchronous)

Compared with the SYNC and NEARSYNC modes, the ASYNC mode results in shorter transaction response times but might cause greater transaction losses if the primary database fails

In ASYNC mode, log writes are considered successful only when the log records have been written to the log files on the primary database and have been delivered to the TCP layer of the primary system's host machine. Because the primary system does not wait for acknowledgement from the standby system, transactions might be considered committed when they are still on their way to the standby database.

A failure on the primary database host machine, on the network, or on the standby database can cause log records in transit to be lost. If the primary database is available, the missing log records can be resent to the standby database when the pair reestablishes a connection. However, if a failover operation is required while there are missing log records, those log records will never reach the standby database, causing the associated transactions to be lost in the failover.

If transactions are lost, the new primary database is not exactly the same as the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database will not be able to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records, the log sequences on the primary and standby databases will be different, and attempts to restart the original primary database as a standby database will fail. Because there is a greater possibility of log records being lost if a failover occurs in asynchronous mode, there is also a greater possibility that the primary database will not be able to rejoin the HADR pair. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the **TAKEOVER HADR** command without specifying the **BY FORCE** parameters. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

SUPERASYNC (super asynchronous)

This mode has the shortest transaction response time but has also the highest probability of transaction losses if the primary system fails. This mode is useful when you do not want transactions to be blocked or experience elongated response times due to network interruptions or congestion.

In this mode, the HADR pair can never be in peer state or disconnected peer state. The log writes are considered successful as soon as the log records have been written to the log files on the primary database. Because the primary database does not wait for acknowledgement from the standby database, transactions are considered committed irrespective of the state of the replication of that transaction.

A failure on the primary database host machine, on the network, or on the standby database can cause log records in transit to be lost. If the primary database is available, the missing log records can be resent to the standby database when the pair reestablishes a connection. However, if a failover operation is required while there are missing log records, those log records will never reach the standby database, causing the associated transactions to be lost in the failover.

If transactions are lost, the new primary database is not exactly the same as the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

Since the transaction commit operations on the primary database are not affected by the relative slowness of the HADR network or the standby HADR server, the log gap between the primary database and the standby database might continue to increase. It is important to monitor the log gap as it is an indirect measure of the potential number of transactions that might be lost should a true disaster occur on the primary system. In disaster recovery scenarios, any transactions committed during the log gap would not be available to the standby database. Therefore, monitor the log gap by using the **hadr_log_gap** monitor element; if it occurs that the log gap is not acceptable, investigate the network interruptions or the relative speed of the standby database node and take corrective measures to reduce the log gap.

If the primary database fails, it is possible that the original primary database will not be able to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records, the log sequences on the primary and standby databases will be different, and attempts to restart the original primary database as a standby database will fail. Because there is a greater probability of log records being lost if a failover occurs in super asynchronous mode, there is also a greater probability that the primary database will not be able to rejoin the HADR pair. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the **TAKEOVER HADR** command without specifying the **BY FORCE** parameter. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

HADR multiple standby databases

The high availability disaster recover (HADR) feature supports multiple standby databases. Using multiple standbys, you can have your data in more than two sites, which provides improved data protection with a single technology.

When you deploy the HADR feature in multiple standby mode, you can have up to three standby databases in your setup. You designate one of these databases as the *principal HADR standby database*; any other standby database is an *auxiliary HADR standby database*. Both types of HADR standbys are synchronized with the HADR primary database through a direct TCP/IP connection, both types support reads on standby, and you can configure both types for time-delayed log replay. In addition, you can issue a forced or non-forced takeover on any standby. There are a couple of important distinctions between the principal and auxiliary standbys, however:

- IBM Tivoli System Automation for Multiplatforms (SA MP) automated failover is supported only for the principal standby. You must issue a takeover manually on one of the auxiliary standbys to make one of them the primary. Before issuing a manual takeover, you should disable SA MP.
- All of the HADR sync modes are supported on the principal standby, but the auxiliary standbys can only be in SUPERASYNC mode.

There are a number of benefits to using a multiple HADR standby setup. Instead of employing the HADR feature to achieve your high availability objectives and another technology to achieve your disaster recovery objectives, you can use HADR for both. You can deploy your principal standby in the same location as the primary. If there is an outage on the primary, the principal standby can take over the primary role within your recovery time objectives. You can also deploy auxiliary standbys in a distant location, which provides protection against a widespread disaster that affects both the primary and the principal standby. The distance, and the potential for network delays between the primary and the auxiliaries, has no effect on activity on the primary because the auxiliaries use SUPERASYNC mode. If a disaster affects the primary and principal standby, you can issue a takeover on either of the auxiliaries. You can configure the other auxiliary standby database to become the new principal standby using the **hadr_target_list** database configuration parameter. However, an auxiliary standby can take over as the primary even if that auxiliary does not have an available standby. For example, if there is an outage on the primary and principal standby, one auxiliary can take over as the primary even if it does not have a corresponding standby. However, if you stop that database after it becomes the new primary, it cannot start again as an HADR primary unless its principal standby is started.

Restrictions for multiple standby databases

There are a number of restrictions that you should be aware of if you are planning to deploy the HADR feature in multiple standby mode.

The restrictions are as follows:

- You can have a maximum of three standby databases: one principal standby and up to two auxiliary standbys.
- Only the principal standby supports all the HADR synchronization modes; all auxiliary standbys will be in SUPERASYNC mode.
- IBM Tivoli System Automation for Multiplatforms (SA MP) support applies only between the primary HADR database and its principal standby.
- The **hadr_target_list** database configuration parameter must be set on all the databases in the multiple standby setup. Each standby must include the primary in its **hadr_target_list** setting.

Initializing HADR in multiple standby mode

Initializing an HADR system in multiple standby mode is similar to single standby mode. The main difference is that you must enable multiple standby mode by setting the **hadr_target_list** database configuration parameter on all the databases in your setup.

About this task

This task covers how to initialize HADR in multiple standby mode. If you want to convert a single standby setup to a multiple standby setup, see “Enabling multiple standby mode on a preexisting HADR setup” on page 856.

Multiple standby mode requires the **hadr_target_list** configuration parameter to be set on all participating databases. This parameter lists the standbys in the scenario when the database becomes a primary. It is required even on a standby. Mutual inclusion is required (that is, if A has B in its target list, B must have A in its target list). This ensures that after a takeover from any standby, the new primary can always keep the old primary as its standby. The first standby that you specify in the target list is designated as the *principal HADR standby database*. Additional standbys are *auxiliary HADR standby databases*. The target list need not always include all participants. As well, there is no requirement for symmetry or reciprocity if there is more than one standby; even if you designate that database A has database B as its principal standby, database B does not have to designate A as its principal standby. Each standby specified in the target list of database A, must also have database A in its target list. Working out the target list for each database is an important step.

As a special case, multiple standby mode can be configured with only one standby. For example, you can configure two databases as primary and standby in multiple standby mode. The behavior is not same as single standby setup because multiple standby behavior such as automated configuration will be in effect and because standby targets can be added or removed dynamically.

Tip: You can perform steps 2 to 4 in a single update on each database.

Procedure

To initialize HADR in multiple standby mode:

1. Create your standby database or databases by using either a restored backup or split mirror. For instructions on how to do this, see "Initializing a standby database" on page 897 or step 2 of "Initializing high availability disaster recovery (HADR)" on page 894.
   a. On the primary, issue the following command:
      ```
      BACKUP DB dbname
      ```
   b. If the database already exists on a standby instance, drop it first for a clean start. Files from the existing database can interfere with HADR operation. For example, left over log files can lead the standby onto a log chain not compatible with the primary. Issue the following command to drop the database:
      ```
      DROP DB dbname
      ```
   c. On each standby instance, issue the following command :
      ```
      RESTORE DB dbname
      ```
2. On each of the databases, set the **hadr_local_host**, **hadr_local_svc**, **hadr_local_svc**, and **hadr_syncmode** configuration parameters:
   ```
   UPDATE DB CFG FOR dbname USING
    HADR_LOCAL_HOST   hostname
    HADR_LOCAL_SVC    servicename
    HADR_SYNCMODE     syncmode
   ```
3. Set the **hadr_target_list** configuration parameter on all of the standbys and the primary:
   ```
   UPDATE DB CFG FOR dbname USING
    HADR_TARGET_LIST  principalhostname:principalservicename|
    auxhostname1:auxservicename1|auxhostname2:auxservicename2
   ```
4. Optional: On all the databases, set the **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters.

   This step is not required because these values are automatically set if you do not set them and are automatically reset if you set them incorrectly. However, explicitly setting them to the correct values makes correct values available immediately. These values are helpful for the IBM Tivoli System Automation for Multiplatforms (SA MP) software, which might require the **hadr_remote_inst** value to construct the resource name.
   - On the primary, set the parameters to the corresponding values on the principal standby by issuing the following command:
     ```
     UPDATE DB CFG FOR  dbname USING
          HADR_REMOTE_HOST   principalhostname
          HADR_REMOTE_SVC    principalservicename
          HADR_REMOTE_INST   principalinstname
     ```
   - On each standby, set the parameters to the corresponding values on the primary by issuing the following command:
     ```
     UPDATE DB CFG FOR  dbname USING
          HADR_REMOTE_HOST   primaryhostname
          HADR_REMOTE_SVC    primaryservicename
          HADR_REMOTE_INST   primaryinstname
     ```
5. Connect to each standby instance.

6. On the standby instance, issue the **START HADR** command with the **AS STANDBY** parameter:

```
START HADR ON DB dbname AS STANDBY
```

7. Connect to the primary instance.
8. On the primary instance, issue the **START HADR** command with the **AS PRIMARY** parameter:

```
START HADR ON DB dbname AS PRIMARY
```

Results

The standby databases start in local catchup state, in which locally available log files are read and replayed. After all local logs have been replayed, the databases enter remote catchup pending state. After the primary starts, the standbys enter remote catchup state, in which log pages are received from the primary and replayed. After all of the log files that are on the disk of the primary database have been replayed on the standbys, what happens depends on the type of what happens next depends on the type of synchronization mode. A principal standby in SUPERASYNC and any auxiliary standby will stay in remote catchup mode. A principal standby with a SYNC, NEARSYNC, or ASYNC mode will enter peer mode.

Enabling multiple standby mode on a preexisting HADR setup

Initializing an HADR system in multiple standby mode is similar to s single standby mode. The main difference is that you must enable multiple standby mode by setting the **hadr_target_list** database configuration parameter on all the databases in your setup.

Before you begin

- Determine the host name or host IP address (to be used for the **hadr_local_host** setting), service name or port number (to be used for the **hadr_local_svc** setting) of all participating databases.
- Determine the target list for each database.
- Determine the synchronization mode and peer window for each database's principal standby in the event that the database becomes the primary.
- Determine the setting for the **hadr_timeout** configuration parameter; this parameter must have the same setting on all databases.
- Determine if there is sufficient network bandwidth between the primary and each standby. Upgrade if necessary.
- Determine if the primary network interface can support outgoing data flow of the additional standbys. Upgrade if needed.

About this task

Multiple standby mode requires the **hadr_target_list** configuration parameter to be set on all participating databases. This parameter lists the standbys in the scenario when the database becomes a primary. It is required even on a standby. Mutual inclusion is required (that is, if A has B in its target list, B must have A in its target list). This ensures that after a takeover from any standby, the new primary can always keep the old primary as its standby. The first standby that you specify in the target list is designated as the *principal HADR standby database*. Additional standbys are *auxiliary HADR standby databases*. The target list need not always include all participants. As well, there is no requirement for symmetry or reciprocity if there is more than one standby; even if you designate that database A has database B as its principal standby, database B does not have to designate A as its principal standby. Each standby specified in the target list of database A, must also have database A in its target list. Working out the target list for each database is an important step.

As a special case, multiple standby mode can be configured with only one standby. For example, you can configure two databases as primary and standby in multiple standby mode. The behavior is not same as single standby setup because multiple standby behavior such as automated configuration will be in effect and because standby targets can be added or removed dynamically.

In this task, you first create and configure the new standbys only. By keeping the original configuration until the final steps, you can keep your primary-standby pair functioning for as long as possible. If you change the original standby's configuration too early, you can break the old HADR pair if the standby is deactivated and reactivated unintentionally to pick up the new configuration.

Procedure

To enable HADR in multiple standby mode:

1. Create any additional standby databases using either a restored backup or split mirror. For instructions on how to do this, see “Initializing a standby database” on page 897 or step 2 of “Initializing high availability disaster recovery (HADR)” on page 894.
   - On the primary:
     ```
     DB2 BACKUP DB dbname
     ```
   - On the standbys:
     ```
     DB2 RESTORE DB dbname
     ```
2. Configure each of the new standby databases as follows:
   a. Set the **hadr_local_host** and **hadr_local_svc** to the TCP address used by the HADR connection.
   b. Set the **hadr_remote_host**, **hadr_remote_svc**, **hadr_remote_inst** configuration parameters to point to the primary database.
   c. Set the **hadr_timeout** configuration, with the same setting on all of the databases.
   d. Set the **hadr_target_list** configuration parameter, as previously planned.
   e. Set the **hadr_syncmode** and **hadr_peer_window** configuration parameters for the principal standby in case this database becomes the primary.
   f. Set any other HADR-specific parameters such as **hadr_spool_limit** or **hadr_replay_delay**, depending on your desired setup.
3. Reconfigure the original standby by following the same instructions as in Step 2.
4. Reconfigure the primary as follows:
   a. Set the **hadr_local_host** and **hadr_local_svc** to the TCP address used by the HADR connection. You might need to make an update if you are using a new network interface card (NIC) to support higher network bandwidth to accommodate more standbys.
   b. Set the **hadr_remote_host**, **hadr_remote_svc**, **hadr_remote_inst** configuration parameters to point to the principal standby database.
   c. Set the **hadr_timeout** configuration, with the same setting as on all of the standby databases.
   d. Set the **hadr_target_list** configuration parameter, as previously planned.

e. Set the **hadr_syncmode** and **hadr_peer_window** configuration parameters, which the principal standby will use.

f. Set any other HADR-specific parameters such as **hadr_spool_limit** or **hadr_replay_delay**, depending on your desired setup.

5. Stop HADR on the primary.

```
STOP HADR ON DB dbname
```

If a primary (in single standby mode) is still running when a new HADR standby is started, the standby is found incompatible and shut down when it attempts to connect to the primary.

6. Deactivate and then reactivate the original standby to pick up the new configuration.
7. Connect to each new standby instance and issue the **START HADR** command with the **AS STANDBY** option.

```
START HADR ON DB dbname AS STANDBY
```

8. Stop HADR on the primary.

```
START HADR ON DB dbname AS PRIMARY
```

Results

All of the standbys should connect to the primary within seconds. You can monitor their status using the **db2pd** command with the -hadr option or the MON_GET_HADR table function.

Modifications to a multiple standby database setup

After your multiple HADR standby setup is up and running, you might want to make additional changes, such as adding or removing auxiliary standby databases or changing the principal standby database designation. You can make these kinds of modifications without causing an outage on your primary database.

Adding auxiliary standbys

There are a few reasons why you might want to add an auxiliary standby:

- To deploy an additional standby for processing read-only workloads
- To deploy an additional standby for time-delayed replay
- To deploy an additional standby for disaster recovery purposes
- To add a standby that was a part of a previously active HADR deployment but was *orphaned* because the **hadr_target_list** configuration parameter for the new primary does not specify that standby

You can add an auxiliary standby only if your HADR deployment is in multiple standby mode. That is, the**hadr_target_list** configuration parameter must already be set to at least one standby.

To add an auxiliary standby to your HADR deployment, update the target list of the primary with the host and port information from the standby. This information corresponds to the settings for the **hadr_local_host** and **hadr_local_svc** parameters on the standby. You must also add the host and port information for the primary to the target list of the new standby.

Tip: Although it is not required, a best practice is to also add the host and port information for the new standby to the target lists of the other standbys in the deployment. You should also specify the host and port information for those standbys in the target list of the new standby. If you do not make these additional updates and one of the other standbys takes over as the new primary, the new standby is rejected as a standby target and is shut down.

Removing auxiliary standbys

The only standbys that you can remove dynamically are auxiliary standbys. If you dynamically remove an auxiliary standby from your multiple standby deployment, there is no effect on normal HADR operations on the primary and the principal standby. To remove an auxiliary standby, issue the **STOP HADR** command on the standby; afterward, you can remove it from the target lists of the primary and any other standby.

Changing the principal standby

You can change the principal standby only if you first stop HADR on the primary database; this does not cause an outage, because you do not have to deactivate the primary.

To change the principal standby, you must stop HADR on the primary database. Then, update the target list of the primary database to list the new principal standby first. If the new principal standby is not already a standby, add the primary database's address to its target list, configure the other HADR parameters, and activate the standby. If it is already a standby, no action is needed.

Tip: Although it is not required, it is a best practice to also add the host and port information for the new principal standby to the target list of the other standby in the deployment. You should also specify the host and port information for that standby in the target list of the new principal standby. If you do not make these additional updates and either one of the standbys takes over as the new primary, the other standby is rejected as a standby target and is shut down.

Database configuration for multiple HADR standby databases

There are a number of considerations for database configuration in a multiple HADR standby setup.

Automatic reconfiguration of HADR parameters

Reconfiguration after HADR starts

In multiple standby mode, the configuration parameters that identify the primary database for the standbys and identify the principal standby for the primary are automatically reset when HADR starts if you did not correctly set them; however, an initial non-NULL value is required. This behavior applies to the following configuration parameters:

- **hadr_remote_host**
- **hadr_remote_inst**
- **hadr_remote_svc**

Tip: Even though this automatic reconfiguration occurs, you should always try to set the correct initial values because that reconfiguration might not take effect until a connection is made between a standby and its primary. In some HADR deployments, those initial values might be needed. For example, if you are using the IBM Tivoli System Automation for Multiplatforms software, the value for the **hadr_remote_inst** configuration parameter is needed to construct a resource name.

Note: If the **DB2_HADR_NO_IP_CHECK** registry variable is set to ON, the **hadr_remote_host** and **hadr_remote_svc** are not automatically updated.

Reconfiguration is predicated on the values of the **hadr_target_list** configuration parameter being correct; if anything is wrong in a target list entry, you must correct it manually.

On the primary, the reconfiguration occurs in the following manner:

- If the values for the **hadr_remote_host** and **hadr_remote_svc** configuration parameters do not match the *host*:*port* pair that is the first entry of the **hadr_target_list** configuration parameter (namely, the principal standby), the **hadr_remote_host** and **hadr_remote_svc** configuration parameters are updated with the values from the target list.
- If the value for the **hadr_remote_inst** configuration parameter does not match the instance name of the principal standby, the correct instance name is copied to the **hadr_remote_inst** configuration parameter for the primary after the principal standby connects to it.

On a standby database, the reconfiguration occurs in the following manner:

- When a standby starts, it attempts to connect to the database that you specified for its **hadr_remote_host**, **hadr_remote_inst**, and **hadr_remote_svc** configuration parameters.
- If the standby cannot connect to the primary, it waits for the primary to connect to it.
- The primary attempts to connect to its standbys using addresses listed in its **hadr_target_list** parameter. After the primary connects to a standby, the **hadr_remote_host**, **hadr_remote_inst**, and **hadr_remote_svc** configuration parameters for the standby are updated with the correct values for the primary.

Reconfiguration during and after a takeover

In a non-forced takeover, the values for the **hadr_remote_host**, **hadr_remote_inst**, and **hadr_remote_svc** configuration parameters on the new primary are automatically updated to its principal standby, and these parameters on the standbys listed in the new primary's **hadr_target_list** are automatically updated to point to the new primary. Any database that is not listed in the new primary's **hadr_target_list** is not updated. Those databases continue to attempt to connect to the old primary and get rejected because the old primary is now a standby. The old primary is guaranteed to be in the new primary's target list because of the requirement of mutual inclusion in the target list.

In a forced takeover, automatic update on the new primary and its standbys (excluding the old primary) work the same way as non-forced takeover. However, automatic update on the old primary does not happen until it is shut down and restarted as a standby for reintegration.

Any database that is not online during the takeover will be automatically reconfigured after it starts. Automatic reconfiguration might not take effect immediately on startup, because it relies on the new primary to periodically contact the standby. On startup, a standby might attempt to connect to the old primary and follow the log stream of the old primary, causing it to diverge from the new primary's log stream and, making that standby unable to pair with the new primary. As a result, you must shut down the old primary before takeover to avoid that kind of *split brain* scenario.

Lack of standby control of the synchronization mode and peer window

In multiple standby mode, only the settings of the **hadr_syncmode** and **hadr_peer_window** configuration parameters of the current primary are relevant. The standby databases either have the settings for those parameters defined by the primary, in the case of the principal standby, or by their role as an auxiliary standby.

Synchronization mode

In multiple standby mode, the setting for the **hadr_syncmode** configuration parameter do not have to be the same on the primary and standby databases. Whatever setting you specify for the **hadr_syncmode** configuration parameter on a standby is considered its *configured synchronization mode*; this setting has relevance only if the standby becomes a primary. The standby is assigned an *effective synchronization mode*. For any auxiliary standby, the effective synchronization mode is always SUPERASYNC. For the principal standby, the effective synchronization mode is the setting for the **hadr_syncmode** configuration parameter for the primary. For a standby, the monitoring interfaces display the effective synchronization mode as the synchronization mode.

Peer window

In multiple standby mode, the setting for the **hadr_peer_window** configuration parameter does not have to be the same on the primary and standby databases. In fact, any setting for the **hadr_peer_window** configuration parameter on the auxiliary standbys is ignored because peer window functionality is incompatible with SUPERASYNC mode. The principal standby uses the peer window setting of the primary, which is applicable only if the value of the **hadr_syncmode** configuration parameter for the standby is SYNC or NEARSYNC, just as with single standby mode.

Rolling upgrades in HADR multiple standby mode

As with HADR single standby mode, you can use a rolling upgrade. The crucial difference is that with multiple standbys you can use this procedure while maintaining HADR protection by keeping a primary and a standby active.

There is always a primary to provide database service and this primary always has at least one standby providing HA and DR protection.

With multiple standbys, you should perform the update or upgrade on all of the standbys before doing so on the primary. This is particularly important if you are updating the fixpack level because HADR does not allow the primary to be at a higher fixpack level than the standby.

The procedure is essentially the same as with single standby mode, except you should perform the upgrade on one database at a time and starting with an auxiliary standby. For example, consider the following HADR setup:

- host1 is the primary
- host2 is the principal standby
- host 3 is the auxiliary standby

For this setup, perform the rolling upgrade or update according to the following sequence:

1. Deactivate host3, make the required changes, activate host3, and start HADR on host3 (as a standby).
2. After host3 is caught up in log replay, deactivate host2, make the required changes, activate host2, and start HADR on host2 (as a standby).
3. After host2 is caught up in log replay and in peer state with host1, issue a takeover on host2.
4. Deactivate host1, make the required changes, activate host1, and start HADR on host1 (as a standby).
5. After host1 is in peer state with host 2, issue a takeover on host1 so that it becomes the primary again and host2 becomes the principal standby again.

High availability disaster recovery (HADR) monitoring in multiple standby mode

HADR multiple standby mode supports the same monitoring interfaces as in single standby mode; however, you should only use the **db2pd** command and the MON_GET_HADR table function because other monitoring interfaces do not give a complete view of all of the standbys.

The information returned by the monitoring interface depends on where it is issued. Monitoring on a standby returns information about that standby and the primary only; no information is provided about any other standbys. Monitoring on the primary returns information about all of the standbys if you are using the **db2pd** command or the MON_GET_HADR table function. Even standbys that are not connected, but are configured in the primary's **hadr_target_list** configuration parameter are displayed. Other interfaces like the **GET SNAPSHOT FOR DATABASE** command report the primary and the principal standby only.

The **db2pd** command and the MON_GET_HADR table function return essentially the same information, but the **db2pd** command does not require reads on standby to be enabled (for reporting from a standby). As well, the **db2pd** command is preferred during takeover because there could be a time window where neither the primary nor the standby allows client connections.

db2pd command

In the following example, the DBA issues the **db2pd** command on a primary database with three standbys. Three sets of data are returned, with each representing a primary-standby log shipping channel. The HADR_ROLE field represents the role of the database to which **db2pd** is issued, so it is listed as PRIMARY in all sets. The HADR_STATE for the two auxiliary standbys (hostS2 and hostS3) is REMOTE_CATCHUP because they automatically run in SUPERASYNC mode (which is also reflected in the **db2pd** output) regardless of their configured setting for **hadr_syncmode**. The STANDBY_ID differentiates the standbys. It is system generated and the ID-to-standby mapping can change from query to query; however, the ID "1" is always assigned to the principal standby.

Note: Fields not relevant to current status might be omitted in the output. For example, in the following output, information about the replay-only window (like start time and transaction count) is not included because the replay-only window is not active.

```
db2pd -db hadr_db -hadr

  Database Member 0 -- Database hadr_db -- Active -- Up 0 days 00:23:17 --
Date 06/08/2011 13:57:23
```

```
                            HADR_ROLE = PRIMARY
                          REPLAY_TYPE = PHYSICAL
                        HADR_SYNCMODE = SYNC
                           STANDBY_ID = 1
                        LOG_STREAM_ID = 0
                           HADR_STATE = PEER
                  PRIMARY_MEMBER_HOST = hostP.ibm.com
                     PRIMARY_INSTANCE = db2inst1
                       PRIMARY_MEMBER = 0
                  STANDBY_MEMBER_HOST = hostS1.ibm.com
                     STANDBY_INSTANCE = db2inst2
                       STANDBY_MEMBER = 0
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
          HEARTBEAT_INTERVAL(seconds) = 30
                HADR_TIMEOUT(seconds) = 120
        TIME_SINCE_LAST_RECV(seconds) = 3
             PEER_WAIT_LIMIT(seconds) = 0
           LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
   LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
                  LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
                  HADR_LOG_GAP(bytes) = 0
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
       STANDBY_RECV_REPLAY_GAP(bytes) = 0
                     PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                     STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
              STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
         STANDBY_RECV_BUF_SIZE(pages) = 16
             STANDBY_RECV_BUF_PERCENT = 0
           STANDBY_SPOOL_LIMIT(pages) = 0
                 PEER_WINDOW(seconds) = 0
             READS_ON_STANDBY_ENABLED = Y
    STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

                            HADR_ROLE = PRIMARY
                          REPLAY_TYPE = PHYSICAL
                        HADR_SYNCMODE = SUPERASYNC
                           STANDBY_ID = 2
                        LOG_STREAM_ID = 0
                           HADR_STATE = REMOTE_CATCHUP
                  PRIMARY_MEMBER_HOST = hostP.ibm.com
                     PRIMARY_INSTANCE = db2inst1
                       PRIMARY_MEMBER = 0
                  STANDBY_MEMBER_HOST = hostS2.ibm.com
                     STANDBY_INSTANCE = db2ins3t
                       STANDBY_MEMBER = 0
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 06/08/2011 13:35:51.724447 (1307565351)
          HEARTBEAT_INTERVAL(seconds) = 30
                HADR_TIMEOUT(seconds) = 120
        TIME_SINCE_LAST_RECV(seconds) = 16
             PEER_WAIT_LIMIT(seconds) = 0
           LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
   LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
                  LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
                  HADR_LOG_GAP(bytes) = 0
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
       STANDBY_RECV_REPLAY_GAP(bytes) = 0
                     PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                     STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
              STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
         STANDBY_RECV_BUF_SIZE(pages) = 16
             STANDBY_RECV_BUF_PERCENT = 0
           STANDBY_SPOOL_LIMIT(pages) = 0
                 PEER_WINDOW(seconds) = 0
             READS_ON_STANDBY_ENABLED = Y

                            HADR_ROLE = PRIMARY
                          REPLAY_TYPE = PHYSICAL
                        HADR_SYNCMODE = SUPERASYNC
                           STANDBY_ID = 3
                        LOG_STREAM_ID = 0
                           HADR_STATE = REMOTE_CATCHUP
                  PRIMARY_MEMBER_HOST = hostP.ibm.com
                     PRIMARY_INSTANCE = db2inst1
                       PRIMARY_MEMBER = 0
                  STANDBY_MEMBER_HOST = hostS3.ibm.com
                     STANDBY_INSTANCE = db2inst3
                       STANDBY_MEMBER = 0
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 06/08/2011 13:46:51.561873 (1307566011)
          HEARTBEAT_INTERVAL(seconds) = 30
                HADR_TIMEOUT(seconds) = 120
        TIME_SINCE_LAST_RECV(seconds) = 6
             PEER_WAIT_LIMIT(seconds) = 0
           LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
   LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
                  LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
                  HADR_LOG_GAP(bytes) = 0
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
       STANDBY_RECV_REPLAY_GAP(bytes) = 0
                     PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                     STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
              STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
         STANDBY_RECV_BUF_SIZE(pages) = 16
             STANDBY_RECV_BUF_PERCENT = 0
           STANDBY_SPOOL_LIMIT(pages) = 0
                 PEER_WINDOW(seconds) = 0
             READS_ON_STANDBY_ENABLED = N
```

MON_GET_HADR table function

In the following example, the DBA calls the **MON_GET_HADR** table function on the primary database with three standbys. Three rows are returned. Each row represents a primary-standby log shipping channel. The HADR_ROLE column represents the role of the database to which the query is issued. Therefore it is PRIMARY on all rows. The HADR_STATE for the two auxiliary standbys (hostS2 and hostS3) is REMOTE_CATCHUP because they automatically run in SUPERASYNC mode regardless of their configured setting for **hadr_syncmode**.

```
db2 "select HADR_ROLE, STANDBY_ID, HADR_STATE, varchar(PRIMARY_MEMBER_HOST,20)
 as PRIMARY_MEMBER_HOST, varchar(STANDBY_MEMBER_HOST,20)
 as STANDBY_MEMBER_HOST from table (mon_get_hadr(NULL))"

  HADR_ROLE  STANDBY_ID HADR_STATE      PRIMARY_MEMBER_HOST  STANDBY_MEMBER_HOST
  ---------  ---------- --------------  -------------------  --------------------
  PRIMARY             1 PEER            hostP.ibm.com        hostS1.ibm.com
  PRIMARY             2 REMOTE_CATCHUP  hostP.ibm.com        hostS2.ibm.com
  PRIMARY             3 REMOTE_CATCHUP  hostP.ibm.com        hostS3.ibm.com

 3 record(s) selected.
```

Takeover in HADR multiple standby mode

When an HADR standby database takes over as the primary database in a multiple standby environment, there are a number of important differences from single standby mode.

With HADR, there are two types of takeover: *role switch* and *failover*. Role switch, sometimes called graceful takeover or non-forced takeover, can be performed only when the primary is available and it switches the role of primary and standby. Failover, or forced takeover, can be performed when the primary is not available. It is commonly used in primary failure cases to make the standby the new primary. The old primary remains in primary role in a forced takeover. Both types of takeover are supported in multiple standby mode, and any of the standby databases can take over as the primary. A crucial thing to remember, though, is that if a standby is not included in the new primary's target list, it is considered to be orphaned and cannot connect to the new primary.

In a takeover, DB2 automatically makes a number of configuration changes for you so that the standbys listed in new primary's target list can connect to the new primary. The **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters are updated on the new primary and listed standbys in the following way:

- **On the new primary:** They refer to the principal standby (the first database listed in the new primary's target list).
- **On the standbys:** They refer to the new primary. When an old primary is reintegrated to become standby, the **START HADR** AS STANDBY command first converts it to a standby. Thus it can also be automatically redirected to the new primary if it is listed in the target list of the new primary.

  Note: Orphaned standbys are not automatically updated in this way. If you want them to join as standbys, you need to ensure they are in the new primary's target list and that they include the new primary in their target lists.

Role switch

Just as in single standby mode, role switch in multiple standby mode guarantees no data is lost between the old primary and new primary. Other standbys configured in the new primary's **hadr_target_list** configuration parameter are automatically redirected to the new primary and continue receiving logs. If you are issuing the **TAKEOVER HADR** command on an auxiliary standby and you have IBM Tivoli System Automation for Multiplatforms (SA MP) configured, you must ensure that you disable SA MP before attempting the takeover. You cannot failback the primary role to the auxiliary primary if SA MP is enabled.

Failover

Just as in single standby mode, if a failover results in any data loss in multiple standby mode (meaning that the new primary does not have all of the data of the old primary), the old and new primary's log streams diverge and the old primary has to be reinitialized. For the other standbys, if a standby received logs from the old primary beyond the diverge point, it has to be reinitialized. Otherwise, it can connect to the new primary and continue log shipping and replay. As a result, it is very important that you check the log positions of all of the standbys and choose the standby with the most data as the failover target. You can query this information using the **db2pd** command or the MON_GET_HADR table function.

Note: Successful automatic reconfiguration of a standby's **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters to point to the new primary does not mean the standby will be accepted to pair with the new primary. It only allows the standby to make a TCP connection to the primary. Upon connection, if DB2 determines that the two databases have diverging log streams, the pairing request will be rejected and the connection closed.

Scenario: Deploying an HADR multiple standby database setup

This scenario describes the planning, configuring, and deploying of an HADR setup for a bank called ExampleBANK. The setup has three standby databases: one principal standby and two auxiliary standbys.

Background

Because banking is a 24x7 business, high availability is crucial to ExampleBANK's technology strategy. In addition, ExampleBANK experienced a close call with a hurricane hitting City A, where its head office is located, so the bank also requires a disaster recovery strategy. High availability disaster recovery (HADR) offers a solution that can help the bank achieve both of these goals with a single technology: HADR multiple standby databases.

ExampleBANK considers the following requirements essential for its HADR solution:

An aggressive recovery time objective
As a bank that offers 24-hour online service, ExampleBANK wants to minimize the time that applications cannot connect to their database.

An aggressive recovery point objective
ExampleBANK cannot tolerate data loss, so the RPO should be as close to 0 as possible.

Near-zero planned downtime
ExampleBANK's database should be available as much as possible, even through planned activities such as upgrades and maintenance.

Data protection through geographic dispersion
As part of its compliance standards, ExampleBANK wants the capability to recover operations at a remote location.

Easy deployment and management
ExampleBANK's overburdened IT department wants a solution that is relatively simple to configure and that has automation capabilities.

As the following scenarios illustrate, using the HADR feature in multiple standby mode helps ExampleBANK meet all these requirements.

Planning for a multiple standby setup

ExampleBANK wants to have both high availability and disaster recovery protection from its HADR setup, so the bank decides to use the maximum number of standbys: three. To achieve the RTO, the bank must have a standby that is in close synchronization with the primary (a standby that uses SYNC or NEARSYNC mode) and is collocated with the primary. It makes the most sense to have this standby be the principal standby because only that standby supports all synchronization modes. Both the primary and the principal standby are located in ExampleBANK's head office in City A and are connected by a LAN.

In addition, to protect the bank's data from being lost because of a disaster, the ExampleBANK DBA chooses to set up two standbys in the bank's regional office in City B. The regional office is connected to the head office in City A by a WAN. The distance between the two cities will not affect the primary because the standbys are auxiliary standbys, which automatically run in SUPERASYNC mode. The DBA can provide additional justification for the costs of these additional databases by setting up one of them to use the reads on standby feature and the other to use the time-delayed replay feature. Also, these standbys can help maintain database availability through a rolling update or maintenance scenario, preventing the loss of HADR protection.

Configuring a multiple standby setup

The ExampleBANK DBA takes a backup of the intended primary database, HADR_DB:

```
DB2 BACKUP DB hadr_db TO backup_dir
```

The DBA then restores the backup onto each of the intended standby hosts by issuing the following command:

```
DB2 RESTORE DB hadr_db FROM backup_dir
```

Tip: For more information about options for creating a standby, see "Initializing a standby database" on page 897.

For the initial setup, the ExampleBANK DBA decides that most of the default configuration settings are sufficient. However, as in a regular HADR setup, the following database configuration parameters must be explicitly set:

- **hadr_local_host**
- **hadr_local_svc**
- **hadr_remote_host**
- **hadr_remote_inst**
- **hadr_remote_svc**

To obtain the correct values for those configuration parameters, the DBA determines the host name, port number, and instance name of the four databases that will be in the HADR setup:

Table 128. Host name, port number, and instance name for databases

Intended role	Host name	Port number	Instance name
Primary	host1	10	dbinst1
Principal standby	host2	40	dbinst2
Auxiliary standby	host3	41	dbinst3
Auxiliary standby	host4.ibm.com	42	dbinst4

On the primary, the settings for the **hadr_remote_host**, **hadr_remote_inst**, and **hadr_remote_svc** configuration parameters correspond to the host name, instance name, and port number of the principal standby. On the standbys, the values of these configuration parameters correspond to the host name, port number, and instance name of the primary. In addition, the DBA uses the host name and port values to set the **hadr_target_list** configuration parameter on all the databases. Also, although it is not required, the DBA adds the information about all the standbys in the setup to the target list of each of the other standbys. For more information about this topic, see "Database configuration for high availability disaster recovery (HADR)" on page 903.

As mentioned earlier, the bank wants the closest possible synchronization between the primary and principal standby, so the DBA sets the **hadr_syncmode** parameter on the primary to SYNC. Although the principal standby will automatically have its effective synchronization mode set to SYNC after it connects to the primary, the DBA still sets the **hadr_syncmode** parameter to SYNC on the principal standby. The reason is that if the principal standby switches role with the primary, the synchronization mode for the new primary and principal standby pair will also be SYNC.

The DBA decides to specify host2, which is in a different city from the auxiliary standbys, as the principal standbys for the auxiliary standbys. If one of the auxiliaries becomes the primary, SUPERASYNC would be a good synchronization mode between the primary and the remotely located host2. Thus DBA sets the **hadr_syncmode** parameter on the auxiliary standbys to SUPERASYNC, although the auxiliary standbys will automatically have their effective synchronization modes set to SUPERASYNC after they connect to the primary. For more information about this topic, see "High Availability Disaster Recovery (HADR) synchronization mode" on page 849.

Finally, the DBA has read about the new HADR delayed replay feature, which can be used to intentionally keep a standby database at a point in time that is earlier than the primary by delaying replay of logs. The DBA decides that enabling this feature would improve ExampleBANK's data protection against errant transactions on the primary. The DBA chooses host4, an auxiliary standby, for this feature, and makes a note that this feature must be disabled before host4 can take over as the primary database. For more information about this topic, see "HADR delayed replay" on page 881.

The DBA issues the following commands to update the configuration parameters on each of the databases:

- On host1 (the primary):

```
DB2 "UPDATE DB CFG FOR hadr_db USING
     HADR_TARGET_LIST  host2:40|host3:41|host4:42
     HADR_REMOTE_HOST  host2
     HADR_REMOTE_SVC   40
     HADR_LOCAL_HOST   host1
     HADR_LOCAL_SVC    10
     HADR_SYNCMODE     sync
     HADR_REMOTE_INST  db2inst2"
```

- On host2 (the principal standby):

```
DB2 "UPDATE DB CFG FOR hadr_db USING
     HADR_TARGET_LIST  host1:10|host3:41|host4:42
     HADR_REMOTE_HOST  host1
     HADR_REMOTE_SVC   10
     HADR_LOCAL_HOST   host2
     HADR_LOCAL_SVC    40
     HADR_SYNCMODE     sync
     HADR_REMOTE_INST  db2inst1"
```

- On host3 (an auxiliary standby):

```
DB2 "UPDATE DB CFG FOR hadr_db USING
     HADR_TARGET_LIST  host2:40|host1:10|host4:42
     HADR_REMOTE_HOST  host1
     HADR_REMOTE_SVC   10
     HADR_LOCAL_HOST   host3
     HADR_LOCAL_SVC    41
     HADR_SYNCMODE     superasync
     HADR_REMOTE_INST  db2inst1"
```

- On host4 (an auxiliary standby):

  ```
  DB2 "UPDATE DB CFG FOR hadr_db USING
       HADR_TARGET_LIST  host2.:40|host1:10|host3:41
       HADR_REMOTE_HOST  host2
       HADR_REMOTE_SVC   10
       HADR_LOCAL_HOST   host4
       HADR_LOCAL_SVC    42
       HADR_SYNCMODE     superasync
       HADR_REMOTE_INST  db2inst1
       HADR_REPLAY_DELAY 86400"
  ```

Finally, the ExampleBANK DBA wants to enable the HADR reads on standby feature for the following reasons:

- To make online changes to some of the HADR configuration parameters on the standbys
- To call the MON_GET_HADR table function on the standbys
- To divert some of the read-only workload from the primary

The DBA updates the registry variables on the standby databases by issuing the following commands on each of host2, host3, and host4:

```
DB2SET DB2_HADR_ROS=ON
DB2SET DB2_STANDBY_ISO=UR
```

Starting the HADR databases

The DBA starts the standby databases first, by issuing the following command on each of host2, host3, and host 4:

```
DB2 START HADR ON DB hadr_db AS STANDBY
```

Next, the DBA starts HADR on the primary database, on host1:

```
DB2 START HADR ON DB hadr_db AS PRIMARY
```

To verify that HADR is up and running, the DBA queries the status of the databases from the primary on host1 by issuing the **db2pd** command, which returns information about all of the standbys:

```
db2pd -db hadr_db -hadr

  Database Member 0 -- Database hadr_db -- Active -- Up 0 days 00:23:17 --
Date 06/08/2011 13:57:23

                             HADR_ROLE = PRIMARY
                           REPLAY_TYPE = PHYSICAL
                         HADR_SYNCMODE = SYNC
                            STANDBY_ID = 1
                         LOG_STREAM_ID = 0
                            HADR_STATE = PEER
                   PRIMARY_MEMBER_HOST = host1
                      PRIMARY_INSTANCE = db2inst1
                        PRIMARY_MEMBER = 0
                   STANDBY_MEMBER_HOST = host2
                      STANDBY_INSTANCE = db2inst2
                        STANDBY_MEMBER = 0
                   HADR_CONNECT_STATUS = CONNECTED
              HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
           HEARTBEAT_INTERVAL(seconds) = 30
                 HADR_TIMEOUT(seconds) = 120
         TIME_SINCE_LAST_RECV(seconds) = 3
```

```
             PEER_WAIT_LIMIT(seconds) = 0
           LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
   LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
                  LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
                  HADR_LOG_GAP(bytes) = 0
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
       STANDBY_RECV_REPLAY_GAP(bytes) = 0
                     PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                     STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
              STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
         STANDBY_RECV_BUF_SIZE(pages) = 16
             STANDBY_RECV_BUF_PERCENT = 0
           STANDBY_SPOOL_LIMIT(pages) = 0
                 PEER_WINDOW(seconds) = 0
             READS_ON_STANDBY_ENABLED = Y
    STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

                            HADR_ROLE = PRIMARY
                          REPLAY_TYPE = PHYSICAL
                        HADR_SYNCMODE = SUPERASYNC
                           STANDBY_ID = 2
                        LOG_STREAM_ID = 0
                           HADR_STATE = REMOTE_CATCHUP
                  PRIMARY_MEMBER_HOST = host1
                     PRIMARY_INSTANCE = db2inst1
                       PRIMARY_MEMBER = 0
                  STANDBY_MEMBER_HOST = host3
                     STANDBY_INSTANCE = db2inst3
                       STANDBY_MEMBER = 0
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 06/08/2011 13:35:51.724447 (1307565351)
          HEARTBEAT_INTERVAL(seconds) = 30
                HADR_TIMEOUT(seconds) = 120
        TIME_SINCE_LAST_RECV(seconds) = 16
             PEER_WAIT_LIMIT(seconds) = 0
           LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
   LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
                  LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
                  HADR_LOG_GAP(bytes) = 0
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
       STANDBY_RECV_REPLAY_GAP(bytes) = 0
                     PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                     STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
              STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
         STANDBY_RECV_BUF_SIZE(pages) = 16
             STANDBY_RECV_BUF_PERCENT = 0
           STANDBY_SPOOL_LIMIT(pages) = 0
                 PEER_WINDOW(seconds) = 0
             READS_ON_STANDBY_ENABLED = Y
    STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

                            HADR_ROLE = PRIMARY
                          REPLAY_TYPE = PHYSICAL
                        HADR_SYNCMODE = SUPERASYNC
                           STANDBY_ID = 3
                        LOG_STREAM_ID = 0
                           HADR_STATE = REMOTE_CATCHUP
                  PRIMARY_MEMBER_HOST = host1
                     PRIMARY_INSTANCE = db2inst1
                       PRIMARY_MEMBER = 0
                  STANDBY_MEMBER_HOST = host4
```

```
                     STANDBY_INSTANCE = db2inst4
                       STANDBY_MEMBER = 0
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 06/08/2011 13:46:51.561873 (1307566011)
          HEARTBEAT_INTERVAL(seconds) = 30
                HADR_TIMEOUT(seconds) = 120
        TIME_SINCE_LAST_RECV(seconds) = 6
             PEER_WAIT_LIMIT(seconds) = 0
           LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
   LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
                  LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
                  HADR_LOG_GAP(bytes) = 0
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
       STANDBY_RECV_REPLAY_GAP(bytes) = 0
                     PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                     STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
              STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
         STANDBY_RECV_BUF_SIZE(pages) = 16
             STANDBY_RECV_BUF_PERCENT = 0
           STANDBY_SPOOL_LIMIT(pages) = 0
                 PEER_WINDOW(seconds) = 0
             READS_ON_STANDBY_ENABLED = Y
    STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N
```

Examples: Takeover in HADR multiple standby mode

This set of examples of takeovers (both forced and unforced) in HADR multiple standby mode is based on a three-standby setup. The purpose of these examples is to show how the multiple standby automatic reconfiguration works in a takeover situation.

- "A principal standby takes over gracefully (role switch)" on page 872
- "An auxiliary standby takes over by force (failover)" on page 873
- "An auxiliary standby takes over by force (failover) in a SA MP environment" on page 875

The initial setup for each of the examples is as follows:

- a primary database (host1)
- a principal standby (host2)
- two auxiliary standbys (host3 and host4)

All of the databases are called hadr_db. The primary and principal standby have their synchronization mode set to SYNC and the standbys have theirs set to SUPERASYNC.

The configuration for each database is shown in Table 129.

Table 129. Configuration values for each HADR database

Configuration parameter	Host1	Host2	Host3	Host4
hadr_target_list	host2:40 | host3:41 | host4:42	host1:10 | host3:41 | host4:42	host2:40 | host1:10 | host4:42	host2:40 | host1:10 | host3:41
hadr_remote_host	host2	host1	host1	host1
hadr_remote_svc	40	10	10	10
hadr_remote_inst	dbinst2	dbinst1	dbinst1	dbinst1

Table 129. Configuration values for each HADR database (continued)

Configuration parameter	Host1	Host2	Host3	Host4
hadr_local_host	host1	host2	host3	host4
hadr_local_svc	10	40	41	42
Configured **hadr_syncmode** (Refers to the explicitly set synchronization mode, which is used if the database becomes a primary)	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective **hadr_syncmode** (Refers to the synchronization mode that is used if the database is currently a standby)	n/a	SYNC	SUPERASYNC	SUPERASYNC

A principal standby takes over gracefully (role switch)

The DBA performs a takeover on the principal standby by issuing the following command on host2:

```
DB2 TAKEOVER HADR ON DB hadr_db
```

After the takeover is completed successfully, host2 becomes the new primary and host1, which is the first entry in the **hadr_target_list** of host2 (as shown in Table 129 on page 871), becomes its principal standby. Their sync mode is SYNC mode because host2 is configured with an **hadr_syncmode** of SYNC. The auxiliary standby targets, host3 and host4, have their **hadr_remote_host** and **hadr_remote_svc** pointing at the old primary, host1, but are automatically redirected to the new primary, host2. In this redirection, host3 and host4 update (persistently) their **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters. They reconnect to host2 as auxiliary standbys, and are told by host2 to use an effective synchronization mode of SUPERASYNC (regardless of what they have locally configured for **hadr_syncmode**). They do not update their settings for **hadr_syncmode** persistently. The configuration for each database is shown inTable 130.

Table 130. Configuration values for each HADR database after a role switch. Rows 3 to 5 in columns 4 and 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1	Host2	Host3	Host4
hadr_target_list	host2:40 | host3:41 | host4:42	host1:10 | host3:41 | host4:42	host2:40 | host1:10 | host4:42	host2:40 | host1:10 | host3:41
hadr_remote_host	host2	host1	**host2**	**host2**
hadr_remote_svc	40	10	**40**	**40**
hadr_remote_inst	dbinst2	dbinst1	**dbinst2**	**dbinst2**
hadr_local_host	host1	host2	host3	host4

Table 130. Configuration values for each HADR database after a role switch (continued). Rows 3 to 5 in columns 4 and 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1	Host2	Host3	Host4
hadr_local_svc	10	40	41	42
Configured **hadr_syncmode**	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective **hadr_syncmode**	SYNC	n/a	SUPERASYNC	SUPERASYNC

Note: A number of values are not updated for the following reasons

- Because host2 already has its **hadr_remote_host** and **hadr_remote_svc** configuration parameters pointing at its principal standby, host1, these values are not updated on host2.
- Because host1 already has its **hadr_remote_host** and **hadr_remote_svc** configuration parameters pointing at the new primary, these values are not updated on host1.
- Because host1's operational synchronization mode is SYNC and host3 and host4's operational synchronization modes are SUPERASYNC, there is no change for the effective synchronization mode.

An auxiliary standby takes over by force (failover)

A widespread power outage in City A results in the primary (host1) becoming unavailable. Normally, the principal standby (host2) which is in SYNC mode would be the best candidate for taking over and becoming the new primary, but the power outage means that host2 is momentarily unavailable as well. The DBA queries the two auxiliary standbys to determine which one has the most log data:

```
db2pd -hadr -db hadr_db | grep 'PRIMARY_LOG_FILE,PAGE,POS|STANDBY_LOG_FILE,PAGE,POS'
```

The DBA determines that host3 is the most up to date (although it is still a little behind in log replay) and picks that host as the new primary:

```
 DB2 TAKEOVER HADR ON DB hadr_db BY FORCE
```

After the takeover is completed successfully, host3 becomes the new primary. Meanwhile, host2 becomes available again. host3 informs host2 and host4 that it is now the primary. On host3, the values for **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** are reconfigured to point to host2, which is the principal standby because it is the first entry in the **hadr_target_list** on host3. On host2, the synchronization mode is reconfigured to SUPERASYNC because that is the setting for **hadr_syncmode** on host3; in addition, the **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** are updated (persistently). host4 is automatically redirected to the new primary, host3. In this redirection, host4 updates (persistently) its **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters. There is no automatic reconfiguration on host1 until it becomes available again. The configuration for each database is shown inTable 131 on page 874.

Table 131. Configuration values for each HADR database after a failover. Rows 3 to 5 in columns 3 to 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1 (unavailable)	Host2	Host3	Host4
hadr_target_list	host2:40 | host3:41 | host4:42	host1:10 | host3:41 | host4:42	host2:40 | host1:10 | host4:42	host2:40 | host1:10 | host3:41
hadr_remote_host	host2	**host3**	**host2**	**host3**
hadr_remote_svc	40	**41**	**40**	**41**
hadr_remote_inst	dbinst2	**dbinst3**	**dbinst2**	**dbinst3**
hadr_local_host	host1	host2	host3	host4
hadr_local_svc	10	40	41	42
Configured **hadr_syncmode**	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective **hadr_syncmode**	n/a	**SUPERASYNC**	n/a	SUPERASYNC

After a short period of time, host1 becomes available. The DBA tries to start host1 as a standby, but because host1 has more logs than were propagated to host3, host1 is rejected as part of the initial handshake with the new primary. The DBA takes a backup of the new primary, restores it to host1, and starts HADR on that host:

```
DB2 BACKUP DB hadr_db

DB2 RESTORE DB hadr_db

DB2 START HADR ON DB hadr_db AS STANDBY
```

As is shown inTable 132, host1 is reconfigured.

Table 132. Configuration values for a reintegrated standby. Various rows in column 2 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1	Host2	Host3	Host4
hadr_target_list	host2:40 | host3:41 | host4:42	host1:10 | host3:41 | host4:42	host2:40 | host1:10 | host4:42	host2:40 | host1:10 | host3:41
hadr_remote_host	**host3**	host3	host2	host3
hadr_remote_svc	**41**	41	40	41
hadr_remote_inst	**dbinst3**	dbinst3	dbinst2	dbinst3
hadr_local_host	host1	host2	host3	host4
hadr_local_svc	10	40	41	42
Configured **hadr_syncmode**	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective **hadr_syncmode**	**SUPERASYNC**	SUPERASYNC	n/a	SUPERASYNC

If the DBA wants to make host1 the primary again, then all that is required is a failback, which will restore the original configuration shown in Table 129 on page 871.

An auxiliary standby takes over by force (failover) in a SA MP environment

This example is similar to the previous one, but HADR has been deployed with IBM Tivoli System Automation for Multiplatforms (SA MP) to automate failover.

A power failure in City A results in the principal standby (host2) becoming unavailable. Following that, there is an outage on the primary (host1). Normally, SA MP, the cluster manager, would automatically fail over to the principal standby (host2), but the power outage means that one of the auxiliary standbys needs to be the takeover target. Failover cannot be automated to auxiliary standbys, so the DBA must do it manually. However, before doing this, the DBA needs to ensure that TSA is disabled so that if host1 or host2 become available, there is no possibility for a *split brain* situation, in which more than one database is operating independently as a primary. To do this, the DBA issues the following command on host1 and host2 (whenever they become available):

```
db2haicu -disable
```

In addition, the DBA needs to keep host1 offline to eliminate the possibility that the old primary will restart if a client connects to it.

The DBA queries the two auxiliary standbys to determine which one has the most log data:

```
db2pd -hadr -db hadr_db | grep 'STANDBY_LOG_FILE,PAGE,POS'
```

The DBA determines that host3 is the most up to date and picks that host as the new primary.

Then, the DBA issues the force takeover on host3:

```
 DB2 TAKEOVER HADR ON DB hadr_db BY FORCE
```

After the takeover is completed successfully, host3 becomes the new primary. Meanwhile, host2 becomes available again. host3 informs host2 and host4 that it is now the primary. On host3, the values for **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** are reconfigured to point to host2, which is the principal standby because it is the first entry in the **hadr_target_list** on host3. On host2, the synchronization mode is reconfigured to SUPERASYNC because that is the setting for **hadr_syncmode** on host3; in addition, the **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** are updated (persistently). host4 is automatically redirected to the new primary, host3. In this redirection, host4 updates (persistently) its **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters. There is no automatic reconfiguration on host1. The configuration for each database is shown inTable 133.

Table 133. Configuration values for each HADR database after a failover. Rows 3 to 5 in columns 3 to 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1 (unavailable)	Host2	Host3	Host4
hadr_target_list	host2:40 | host3:41 | host4:42	host1:10 | host3:41 | host4:42	host2:40 | host1:10 | host4:42	host2:40 | host1:10 | host3:41
hadr_remote_host	host2	**host3**	**host2**	**host3**
hadr_remote_svc	40	**41**	**40**	**41**
hadr_remote_inst	dbinst2	**dbinst3**	**dbinst2**	**dbinst3**

Table 133. Configuration values for each HADR database after a failover (continued). Rows 3 to 5 in columns 3 to 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1 (unavailable)	Host2	Host3	Host4
hadr_local_host	host1	host2	host3	host4
hadr_local_svc	10	40	41	42
Configured **hadr_syncmode**	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective **hadr_syncmode**	n/a	**SUPERASYNC**	n/a	SUPERASYNC

HADR reads on standby feature

You can use the reads on standby capability to run read-only operations on the standby database in your High Availability and Disaster Recovery (HADR) solution. Read operations running on a standby do not affect the standby's main role of replaying logs shipped from the primary database.

The reads on standby feature reduces the total cost of ownership of your HADR setup. This expanded role of the standby database allows you to utilize the standby in new ways, such as running some of the workload that would otherwise be running on your primary database. This, in turn frees up the primary for additional workloads.

Read and write clients continue to connect to the primary database; however read clients can also connect to the read-enabled standby, or *active standby*, as long as it is not in the local catchup state or the replay-only window. An active standby's main role is still to replay logs shipped from the primary. As a result, the data on the standby should be virtually identical to the data on the primary. In the event of a failover, any user connections to the standby will be terminated while the standby takes over as the new primary database.

All types of read queries, including scrollable and non-scrollable cursors, are supported on the standby. Read capability is supported in all four HADR synchronization modes (SYNC, NEARSYNC, ASYNC, and SUPERASYNC) and in all HADR states except local catchup.

Enabling reads on standby

You can enable the reads on standby feature on your High Availability and Disaster Recovery (HADR) standby database using the **DB2_HADR_ROS** registry variable.

Before you begin

It is recommended that database configuration parameter **logindexbuild** be set to ON. This will prevent a performance impact from query access plans avoiding any invalid indexes.

It is also recommended that you use a virtual IP when you have reads on standby enabled. Client reroute does not differentiate between writable databases (primary and standard databases) and read-only databases (standby databases). Configuring client reroute between the primary and standby might route applications to the database on which they are not intended to be run.

Procedure

1. Set the **DB2_HADR_ROS** registry variable to ON.
2. Set up and initialize the primary and standby databases for HADR. Refer to "Initializing high availability disaster recovery (HADR)" on page 894.

Results

Your standby database is now considered an *active standby*, meaning that it will accept read-only workloads.

What to do next

You can now utilize your standby database as you see fit, such as configuring some of your read-only workload to run on it.

To enable your applications to maintain access to your standby database, follow the steps described in the "Continuous access to Read on Standby databases using Virtual IP addresses" white paper.

Data concurrency on the active standby database

Changes on the HADR primary database may not necessarily be reflected on the HADR active standby database. Uncommitted changes on the primary may not replicate to the standby until the primary flushes, or sends, its logs to disk.

Logs are only guaranteed to be flushed to disk-and, therefore sent to the standby-after they have been committed. Log flushes can also be triggered by undeterministic conditions such as a log buffer full situation. As a result, it is possible for uncommitted changes on the primary to remain in the primary's log buffer for a long time. Because the logger avoids flushing partial pages, this situation may particularly affect small uncommitted changes on the primary.

If your workload running on the standby requires the data to be virtually identical to the data on the primary, you should consider committing your transactions more frequently.

Isolation level on the active standby database

The only isolation level that is supported on an active standby database (an HADR standby database that is read enabled) is Uncommitted Read (UR). If the isolation level requested by an application, statement, or sub-statement is higher than UR, an error will be returned (SQL1773N Reason Code 1).

If you require an isolation level other than UR, consider using the HADR primary instead of the standby for this application. If you simply want to avoid receiving this message, set the **DB2_STANDBY_ISO** registry variable to UR. When **DB2_STANDBY_ISO** is set to UR, the isolation level will be silently coerced to UR. This setting takes precedence over all other isolation settings such as statement isolation and package isolation.

Replay-only window on the active standby database

When an HADR active standby database is replaying DDL log records or maintenance operations, the standby enters the *replay-only window*. When the standby is in the replay-only window, existing connections to the standby are terminated and new connections to the standby are blocked (SQL1776N).

New connections are allowed on the standby after the replay of all active DDL or maintenance operations has completed.

The only user connections that can remain active on a standby in the replay-only window are connections that are executing **DEACTIVATE DATABASE** or **TAKEOVER** commands. When applications are forced off at the outset of the replay-only window, an error is returned (SQL1224N). Depending on the number of readers connected to the active standby, there may be a slight delay before the DDL log records or maintenance operations are replayed on the standby.

There are a number of DDL statements and maintenance operations that, when run on the HADR primary, will trigger a replay-only window on the standby. The following lists are not exhaustive.

DDL statements

- CREATE, ALTER, or DROP TABLE (except DROP TABLE for DGTT)
- CREATE GLOBAL TEMP TABLE
- TRUNCATE TABLE
- RENAME TABLE
- RENAME TABLESPACE
- CREATE, DROP, or ALTER INDEX
- CREATE or DROP VIEW
- CREATE, ALTER, or DROP TABLESPACE
- CREATE, ALTER, or DROP BUFFER POOL
- CREATE, ALTER, or DROP FUNCTION
- CREATE, ALTER, or DROP PROCEDURE
- CREATE or DROP TRIGGER
- CREATE, ALTER, or DROP TYPE
- CREATE, ALTER, or DROP ALIAS
- CREATE or DROP SCHEMA
- CREATE, ALTER, or DROP METHOD
- CREATE, ALTER, or DROP MODULE
- CREATE, ALTER, or DROP NICKNAME
- CREATE, ALTER, or DROP SEQUENCE
- CREATE, ALTER, or DROP WRAPPER
- CREATE, ALTER, or DROP FUNCTION MAPPING
- CREATE or DROP INDEX EXTENSION
- CREATE or DROP INDEX FOR TEXT
- CREATE or DROP EVENT MONITOR
- CREATE, ALTER, or DROP SECURITY LABEL
- CREATE, ALTER, or DROP SECURITY LABEL COMPONENT
- CREATE, ALTER, or DROP SECURITY POLICY
- CREATE or DROP TRANSFORM
- CREATE, ALTER, or DROP TYPE MAPPING
- CREATE, ALTER, or DROP USER MAPPING
- CREATE or DROP VARIABLE
- CREATE, ALTER, or DROP WORKLOAD
- GRANT USAGE ON WORKLOAD

- REVOKE USAGE ON WORKLOAD
- CREATE, ALTER, or DROP SERVICE CLASS
- CREATE, ALTER, or DROP WORK CLASS SET
- CREATE, ALTER, or DROP WORK ACTION SET
- CREATE, ALTER, or DROP THRESHOLD
- CREATE, ALTER, or DROP HISTOGRAM TEMPLATE
- AUDIT
- CREATE, ALTER, or DROP AUDIT POLICY
- CREATE or DROP ROLE
- CREATE, ALTER, or DROP TRUSTED CONTEXT
- REFRESH TABLE
- SET INTEGRITY

Maintenance operations

- Classic, or offline, reorg
- Inplace, or online, reorg
- Index reorg (indexes all, individual index)
- MDC and ITC reclaim reorg
- Load
- Bind or rebind
- db2rbind
- Runstats
- Table move
- Auto statistics
- Auto reorg
- Real Time Statistics

Other operation or actions

- Automatic Dictionary Creation for tables with COMPRESS YES attribute
- Asynchronous Index Cleanup on detached table partition
- Implicit rebind
- Implicit index rebuild
- Manual update of statistics.
- Deferred MDC rollout
- Asynchronous Index cleanup after MDC rollout
- Reuse of a deleted MDC or ITC block on insert into MDC or ITC table
- Asynchronous background processes updating catalog tables SYSJOBS and SYSTASKS for inserting, updating, and deleting tasks

Monitoring the replay-only window

You can monitor the replay-only window using the **db2pd** command with the **-hadr** option (on either the standby or the primary) or the MON_GET_HADR table function (from the primary). The standby's status, including information about the replay-only window, is sent to the primary on every heartbeat.

There are three pertinent elements to monitor:

- **STANDBY_REPLAY_ONLY_WINDOW_ACTIVE**, which indicates whether DDL or maintenance-operation replay is in progress on the standby. Normally, the value is N, but when the replay-only window is active, the value is Y.
- **STANDBY_REPLAY_ONLY_WINDOW_START**, which indicates the time at which the current replay-only window (if there is one) became active.
- **STANDBY_REPLAY_ONLY_WINDOW_TRAN_COUNT**, which indicates the total number of existing uncommitted DDL or maintenance transactions executed so far in the current replay-only window (if there is one).

To use the table function, issue something similar to the following query on the primary:

```
select STANDBY_ID, STANDBY_REPLAY_ONLY_WINDOW_ACTIVE, STANDBY_REPLAY_ONLY_WINDOW_START,
 STANDBY_REPLAY_ONLY_WINDOW_TRAN_COUNT from table (mon_get_hadr(NULL))
```

Here is an example using the **db2pd** command on a standby that is currently in a replay-only window:

```
db2pd -hadr db HADRDB

  Database Member 0 -- Database HADRDB -- Active -- Up 0 days 00:23:17 -- Date 06/08/2011 13:57:23

                              HADR_ROLE = STANDBY
                            REPLAY_TYPE = PHYSICAL
                          HADR_SYNCMODE = NEARSYNC
                             STANDBY_ID = 1
                          LOG_STREAM_ID = 0
                             HADR_STATE = PEER
                    PRIMARY_MEMBER_HOST = hostP.ibm.com
                       PRIMARY_INSTANCE = db2inst
                         PRIMARY_MEMBER = 0
                    STANDBY_MEMBER_HOST = hostS1.ibm.com
                       STANDBY_INSTANCE = db2inst
                         STANDBY_MEMBER = 0
                    HADR_CONNECT_STATUS = CONNECTED
               HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
            HEARTBEAT_INTERVAL(seconds) = 25
                  HADR_TIMEOUT(seconds) = 120
          TIME_SINCE_LAST_RECV(seconds) = 3
               PEER_WAIT_LIMIT(seconds) = 0
             LOG_HADR_WAIT_CUR(seconds) = 0.000
      LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
     LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
                    LOG_HADR_WAIT_COUNT = 82
  SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
  SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
              PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
              STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
                    HADR_LOG_GAP(bytes) = 0
       STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
         STANDBY_RECV_REPLAY_GAP(bytes) = 0
                       PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                       STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
           STANDBY_RECV_BUF_SIZE(pages) = 16
               STANDBY_RECV_BUF_PERCENT = 0
             STANDBY_SPOOL_LIMIT(pages) = 0
                   PEER_WINDOW(seconds) = 0
               READS_ON_STANDBY_ENABLED = Y
      STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = Y
       STANDBY_REPLAY_ONLY_WINDOW_START = 06/08/2011 13:50:23
  STANDBY_REPLAY_ONLY_WINDOW_TRAN_COUNT = 5
```

Recommendations for minimizing the impact of the replay-only window

Because replay operations on an HADR standby take priority over readers, frequent read-only windows can be disruptive to readers connected to or attempting to connect to the standby. To avoid or minimize this impact, consider the following recommendations:

- Run DDL and maintenance operations during a scheduled maintenance window, preferably at off-peak hours.
- Run DDL operations collectively rather than in multiple groups.
- Run **REORG** or **RUNSTATS** only on the required tables instead of all tables.
- Terminate applications on the active standby using the **FORCE APPLICATION** command with the **ALL** option before running the DDL or maintenance operations on the primary. Monitor the replay-only window to determine when it is inactive, and redeploy the applications on the standby.

HADR delayed replay

HADR delayed replay helps prevent data loss due to errant transactions. To implement HADR delayed replay, set the **hadr_replay_delay** database configuration parameter on the HADR standby database.

Delayed replay intentionally keeps the standby database at a point in time that is earlier than that of the primary database by delaying replay of logs on that standby. If an errant transaction is executed on the primary, you have until the configured time delay has elapsed to take action to prevent the errant transaction from being replayed on the standby. To recover the lost data, you can either copy this data back to the primary, or you can have the standby take over as the new primary database.

Delayed replay works by comparing timestamps in the log stream, which is generated on the primary, and the current time of the standby. As a result, it is important to synchronize the clocks of the primary and standby databases. Transaction commit is replayed on the standby according to the following equation:

```
(current time on the standby - value of the hadr_replay_delay configuration parameter) >=
 timestamp of the committed log record
```

You should set the **hadr_replay_delay** database configuration parameter to a large enough value to allow time to detect and react to errant transactions on the primary.

You can use this feature in either single standby mode or multiple standby mode. In multiple standby mode, typically one or more standbys stays current with the primary for high availability or disaster recovery purposes, and one standby is configured with delayed replay for protection against errant transactions. If you use this feature in single standby mode, you should not enable IBM Tivoli System Automation for Multiplatforms because the takeover will fail.

There are several important restrictions for delayed replay:

- You can set the **hadr_replay_delay** configuration parameter only on a standby database.

- A **TAKEOVER** command on a standby with replay delay enabled will fail. You must first set the **hadr_replay_delay** configuration parameter to 0 and then deactivate and reactivate the standby to pick up the new value, and then issue the **TAKEOVER** command.
- The delayed replay feature is supported only in SUPERASYNC mode. Because log replay is delayed, a lot of unreplayed log data might accumulate on the standby, filling up receive buffer and spool (if configured). In other synchronization modes, this would cause the primary to be blocked.

  The objective of this feature is to protect against application error. If you want to use this feature and ensure that there is no data loss in the event of a primary failure, consider a multiple standby setup with a more synchronous setting on the principal standby.

Recommendations

Delayed replay and disaster recovery

Consider using a small delay if you are using the standby database for disaster recovery purposes and errant transaction protection.

Delayed replay and the HADR reads on standby feature

Consider using a small delay if you are using the standby database for reads on standby purposes, so that reader sessions can see more up-to-date data. Additionally, because reads on standby runs in “uncommitted read” isolation level, it can see applied, but not yet committed changes that are technically still delayed from replay. These uncommitted transactions can be rolled back in errant transaction recovery procedure when you roll forward the standby to the PIT that you want and then stop.

Delayed replay and log spooling

If you enable delayed replay, it is recommended that you also enable log spooling by setting the **hadr_spool_limit** database configuration parameter. Because of the intentional delay, the replay position can be far behind the log receive position on the standby. Without spooling, log receive can only go beyond replay by the amount of the receive buffer. With spooling, the standby can receive many more logs beyond the replay position, providing more protection against data loss in case of primary failure. Note that in either case, because of the mandatory SUPERASYNC mode, the primary won't be blocked by the delayed replay.

Recovering data by using HADR delayed replay

Using the HADR time-delayed replay feature, you can recover data that was lost because of an errant transaction on the primary database by stopping HADR on a standby before that transaction is replayed.

Before you begin

Delayed replay must have already been enabled for your standby database.

If log replay on the standby, indicated by STANDBY_REPLAY_LOG_TIME, has passed the commit time for the errant transaction on the standby, you cannot recover the data using the following procedure. You can determine the STANDBY_REPLAY_LOG_TIME by using the **db2pd** command with the **-hadr** parameter or the MON_GET_HADR table function.

Restriction: A standby database for which you set the **hadr_replay_delay** configuration parameter cannot take over as a primary; you must first disable delayed replay on that standby.

Procedure

To recover from an errant transaction, perform the following steps on the standby on which you enabled delayed replay:

1. Verify the timing:
   a. Ensure that standby has not yet replayed the transaction. The STANDBY_REPLAY_LOG_TIME value must not have reached the errant transaction commit time.
   b. Ensure that the standby has received the relevant logs. The STANDBY_LOG_TIME value, which indicates logs received, must have reached a PIT before the errant transaction commit time, but close to the errant transaction commit time. This will be the rollforward PIT used in step 3. If the standby has not yet received enough log files, you can wait until more logs are shipped over, but you run the risk of the replay time reaching the errant transaction time. For example, if the delay is 1 hour, you should stop HADR no later than 50 minutes after the errant transaction time (allowing a 10-minute safety margin), even if log shipping has yet not reached the PIT that you want.

      Alternatively, if a shared log archive is available and the logs are already archived, then there is no need to wait. If the logs are not archived yet, the logs can be archived using the **ARCHIVE LOG** command. Otherwise, the user can manually copy complete log files from the primary to the time-delayed standby (the overflow log path is preferred, otherwise, use the log path). For these alternate methods, deactivate the standby first to avoid interference with standby log shipping and replay.

   You can determine these times by issuing `db2pd -db` *dbname* `-hadr` or by enabling the reads on standby feature on the standby and then issuing the following query, which uses the MON_GET_HADR table function:

   ```
   DB2 "select HADR_ROLE, STANDBY_ID, STANDBY_LOG_TIME, STANDBY_REPLAY_LOG_TIME,
   varchar(PRIMARY_MEMBER_HOST,20) as PRIMARY_MEMBER_HOST,
   varchar(STANDBY_MEMBER_HOST,20) as STANDBY_MEMBER_HOST
   from table (mon_get_hadr(NULL))"
   ```
2. Stop HADR on the standby database:

   ```
    DB2 STOP HADR ON DATABASE dbname
   ```
3. Roll forward the standby to the PIT that you want and then stop:

   ```
   DB2 ROLLFORWARD DB dbname to time-stamp and STOP
   ```
4. Use one of the following approaches:
   - Restore the lost data on the primary:
     a. Copy the affected data from the standby and send it back to the primary.

        If the errant transaction dropped a table, you could export it on the standby and import it to the primary. If the errant transaction deleted rows from a table, you could export the table on the standby and use an import replace operation on the primary.
     b. Reinitialize the delayed-replay standby because its log stream has diverged from the primary's. No action is needed on any other standbys because they continue to follow the primary and any data repair on the primary is also replicated to them.

c. Restore the database using a backup image taken on the primary. The image can be one taken at any time.

d. Remove all log files in standby log path. This step is important. The **ROLLFORWARD**... STOP command in step 3 made the database log stream diverge from the primary. If the files are left alone, the newly restored database would follow that log stream and also diverge from the primary. Alternatively, you can drop the database before the restore for a clean start, but then you will also lose the current configuration including HADR configuration.

e. Issue the **START HADR** command with the AS STANDBY option on the database. The database should then activate and connect to the primary.

- Have the standby with the intact data become the primary:

  a. Shut down the old primary to avoid split brain

  b. On the delayed-replay database, set the **hadr_replay_delay** configuration parameter to 0. Reconfigure the other parameters like **hadr_target_list** if needed. Then run **START HADR** command with the AS PRIMARY BY FORCE options on the database to convert it to the new primary. Use the BY FORCE option because there is no guarantee that the configured principal standby (which could be the old primary) will be able to connect.

  c. Redirect clients to the new primary.

  d. The other standbys will be automatically redirected to the new primary. However, if a standby received logs from the old primary beyond the point where old and new primary diverge (the PIT used in step 3), it will be rejected by the new primary. If this happens, reinitialize this standby using the same procedure as reinitializing the old primary.

  e. Reinitialize the old primary because its log stream has diverged from the new primary's.

  f. Restore database using a backup image taken on the new primary, or taken on the old primary before the PIT used in step 3.

  g. Remove all log files in the log path. If you do not do this, the newly restored database will follow the old primary's log stream and diverge from the new primary. Alternatively, you can drop the database before the restore for a clean start, but then you also lose the current configuration including HADR configuration.

  h. Issue the **START HADR** command with the AS STANDBY option on the database. The database should then activate and connect to the primary.

Performing rolling updates in a DB2 High Availability Disaster Recovery (HADR) environment

Use this procedure in a high availability disaster recovery (HADR) environment when you upgrade software or hardware, update your DB2 database product software, or change database configuration parameters.

This procedure keeps database service available throughout the rolling update process, with only a momentary service interruption when processing is switched from one database to the other.With multiple standbys, you can provide continued HA and DR protection throughout the rolling update process.

Before you begin

Review the system requirements for HADR. See "System requirements for DB2 high availability disaster recovery (HADR)" on page 887.

The HADR pair should be in peer state before starting the rolling update.

Note: All DB2 fix pack updates, hardware upgrades, and software upgrades should be implemented in a test environment before being applied to your production system.

About this task

Use this procedure to perform a rolling update on your DB2 database system and update the DB2 database product software from one modification level to another. For example, applying a fix pack to a DB2 database product software. During rolling updates, the modification level or fix pack level of the standby database can be later than that of the primary database while testing the new level. However, you should not keep this configuration for an extended period to reduce the risk of using features that might be incompatible between the levels. The primary and standby databases will not connect to each other if the modification level of the DB2 database product software for the primary database is later than that of the standby database.

A rolling update cannot be used to upgrade from an earlier version to a later version of a DB2 database product software. For example, you cannot use this procedure to upgrade a DB2 database product from Version 9.7 to Version 10.1.

You cannot use this procedure to update the DB2 HADR configuration parameters. Updates to HADR configuration parameters should be made separately. Because HADR requires the parameters on the primary and standby to be the same, this might require both the primary and standby databases to be deactivated and updated at the same time.

Procedure

To perform a rolling update in an HADR environment:

1. Update the standby database by issuing the following steps:
   a. Use the **DEACTIVATE DATABASE** command to shut down the standby database.
   b. If necessary, shut down the instance on the standby database.
   c. Change one or more of the following: the software, the hardware, or the DB2 configuration parameters.

      Note: You cannot change any HADR configuration parameters when performing a rolling update.
   d. If necessary, restart the instance on the standby database.
   e. Use the **ACTIVATE DATABASE** command to restart the standby database.
   f. Ensure that HADR enters peer state. Use the MON_GET_HADR table function (on the primary or a read-enabled standby) or the **db2pd** command with the `-hadr` option to check this.
2. Switch the roles of the primary and standby databases:
   a. Issue the **TAKEOVER HADR** command on the standby database.
   b. Direct clients to the new primary database. This can be done using automatic client reroute.

      Note: Because the standby database takes over as the primary database, the new primary database is now updated. If you are applying a DB2 fix pack, the **TAKEOVER HADR** command changes the role of the original primary database to standby database. However, the command does not let the new standby database connect to the newly updated primary database. Because the new standby database uses an older version of the DB2 database system, it might not understand the new log records generated by the updated primary database, and it will be shut down. In order for the new standby database to reconnect with the new primary database (that is, for the HADR pair to reform), the new standby database must also be updated.

3. Update the original primary database (which is now the standby database) using the same procedure as in step 1 on page 885. When you have done this, both databases are updated and connected to each other in HADR peer state. The HADR system provides full database service and full high availability protection.
4. Optional: To enable the HADR reads on standby feature during the rolling update perform the following steps to ensure the consistency of the internal DB2 packages on the standby database before read operations are introduced. The binding of internal DB2 packages occurs at first connection time, and can complete successfully only on the primary database.
   a. Enable the HADR reads on standby feature on the standby database as follows:
      1) Set the **DB2_HADR_ROS** registry variable to `ON` on the standby database.
      2) Use the **DEACTIVATE DATABASE** command to shut down the standby database.
      3) Restart the instance on the standby database.
      4) Use the **ACTIVATE DATABASE** command to restart the standby database.
      5) Ensure that HADR enters peer state. Use the MON_GET_HADR table function (on the primary or a read-enabled standby) or the **db2pd** command with the `-hadr` option to check this.
   b. Switch the roles of the primary and standby database as follows:
      1) Issue the **TAKEOVER HADR** command on the standby database.
      2) Direct clients to the new primary database.
   c. Repeat the same procedure in substep a to enable the HADR reads on standby feature on the new standby database.
5. Optional: If did not perform step 4 and you want to return to your original configuration, switch the roles of the primary and standby database as you did in step 2 on page 885.
6. Optional: In an HADR environment, run **db2updv10** only on the primary database. After running the **db2updv10** command, you might have to restart the database for changes from **db2updv10** command to take effect. To perform a restart:
   a. Restart the standby database by deactivating and reactivating it. The standby database is restarted to prevent the disruption of primary database service.
      1) Run the following command on the standby database:

         ```
         DEACTIVATE
         db dbname
         ```

         where *dbname* is the name of the standby database.
      2) Run the following command on the standby database:

         ```
         ACTIVATE
         db dbname
         ```

         where *dbname* is the name of the standby database.

b. Switch the roles of the primary and standby databases:
   1) Run the following command on the standby database:

      ```
      TAKEOVER
      hadr on db dbname
      ```

      where *dbname* is the name of the standby database.
   2) Direct clients to the new primary database.

      Note: The databases have switched roles. The primary database was previously the standby database and the standby database was previously the primary database.
c. Restart the standby database (formerly the primary database), using the same method as in Step 1.
d. Switch the roles of the primary and standby databases to return the database to their original roles. Switch the roles using the same method as in step 2.

High availability disaster recovery (HADR) support

Consider system requirements and feature limitations when you design your high availability database solution.

System requirements for DB2 high availability disaster recovery (HADR)

To achieve optimal performance with high availability disaster recovery (HADR), ensure that your system meets the following requirements for hardware, operating systems, and for the DB2 database system.

Recommendation: For better performance, use the same hardware and software for the system where the primary database resides and for the system where the standby database resides. If the system where the standby database resides has fewer resources than the system where the primary database resides, it is possible that the standby database will be unable to keep up with the transaction load generated by the primary database. This can cause the standby database to fall behind or the performance of the primary database to degrade. In a failover situation, the new primary database should have the resources to service the client applications adequately.

If you enable reads on standby and use the standby database to run some of your read-only workload, ensure that the standby has sufficient resources. An active standby requires additional memory and temporary table space usage to support transactions, sessions, and new threads as well as queries that involve sort and join operations.

Hardware and operating system requirements

Recommendation: Use identical host computers for the HADR primary and standby databases. That is, they should be from the same vendor and have the same architecture.

The operating system on the primary and standby databases should be the same version, including patches. When the rolling update procedure is used to upgrade the operating system, the operating system versions can be different on the primary and standby during the procedure. To minimize risks, plan ahead to have the procedure completed in a short time and try it out first in a test environment

A TCP/IP interface must be available between the HADR host machines, and a high-speed, high-capacity network is recommended.

DB2 database requirements

The versions of the database systems for the primary and standby databases must be identical; for example, both must be either version 8 or version 9. During rolling updates, the modification level (for example, the fix pack level) of the database system for the standby database can be later than that of the primary database for a short while to test the new level. However, you should not keep this configuration for an extended period of time. The primary and standby databases will not connect to each other if the modification level of the database system for the primary database is later than that of the standby database. In order to use the reads on standby feature, both the primary and the standby databases need to be Version 9.7 Fix Pack 1.

The DB2 database software for both the primary and standby databases must have the same bit size (32 or 64 bit). Table spaces and their containers must be identical on the primary and standby databases. Properties that must be identical include the table space type (DMS or SMS), table space size, container path, container size, and container file type (raw device or file system). The amount of space allocated for log files should also be the same on both the primary and standby databases.

When you issue a table space statement on the primary database, such as CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE, it is replayed on the standby database. You must ensure that the devices involved are set up on both of the databases before you issue the table space statement on the primary database.

The primary and standby databases do not require the same database path. If relative container paths are used, the same relative path might map to different absolute container paths on the primary and standby databases.

Storage groups are fully supported by HADR, including replication of the CREATE STOGROUP, ALTER STOGROUP and DROP STOGROUP statements. Similar to table space containers, the storage paths must exist on both primary and standby.

The primary and standby databases must have the same database name. This means that they must be in different instances.

Redirected restore is not supported. That is, HADR does not support redirecting table space containers. However, database directory and log directory changes are supported. Table space containers created by relative paths will be restored to paths relative to the new database directory.

Buffer pool requirements

Since buffer pool operations are also replayed on the standby database, it is important that the primary and standby databases have the same amount of memory. If you are using reads on standby, you will need to configure the buffer pool on the primary so that the active standby can accommodate log replay and read applications.

Installation and storage requirements for high availability disaster recovery (HADR)

To achieve optimal performance with high availability disaster recovery (HADR), ensure that your system meets the following installation and storage requirements.

Installation requirements

For HADR, instance paths should be the same on the primary and the standby databases. Using different instance paths can cause problems in some situations, such as if an SQL stored procedure invokes a user-defined function (UDF) and the path to the UDF object code is expected to be on the same directory for both the primary and standby server.

Storage requirements

Storage groups are fully supported by HADR, including replication of the CREATE STOGROUP, ALTER STOGROUP and DROP STOGROUP statements. Similar to table space containers, the storage path must exist on both primary and standby. Symbolic links can be used to create identical paths. The primary and standby databases can be on the same computer. Even though their database storage starts at the same path, they do not conflict because the actual directories used have instance names embedded in them (since the primary and standby databases must have the same database name, they must be in different instances). The storage path is formulated as `storage_path_name/inst_name/dbpart_name/db_name/tbsp_name/container_name`.

Table spaces and their containers must be identical on the primary and standby databases. Properties that must be identical include: the table space type (DMS or SMS), table space size, container path, container size, and container file type (raw device or file system). Storage groups and their storage paths must be identical. This includes the path names and the amount of space on each that is devoted to each storage group. The amount of space allocated for log files should also be the same on both the primary and standby databases.

When you issue a table space statement on the primary database, such as CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE, it is replayed on the standby database. You must ensure that the devices involved are set up on both of the databases before you issue the table space statement on the primary database.

If the table space setup is not identical on the primary and standby databases, log replay on the standby database might encounter errors such as OUT OF SPACE or TABLE SPACE CONTAINER NOT FOUND. Similarly, if the storage groups's storage path setup is not identical on the primary and standby databases, log records associated with the CREATE STOGROUP or ALTER STOGROUP are not be replayed. As a result, the existing storage paths might prematurely run out of space on the standby system and automatic storage table spaces are not be able to increase in size. If any of these situations occurs, the affected table space is put in rollforward pending state and is ignored in subsequent log replay. If a takeover operation occurs, the table space is not available to applications.

If the problem is noticed on the standby system prior to a takeover then the resolution is to re-establish the standby database while addressing the storage issues. The steps to do this include:

- Deactivating the standby database.
- Dropping the standby database.

- Ensuring the necessary file systems exist with enough free space for the subsequent restore and rollforward.
- Restoring the database at the standby system using a recent backup of the primary database (or, reinitialize using split mirror or flash copy with the **db2inidb** command). Storage group storage paths should not be redefined during the restore. Also, table space containers should not be redirected as part of the restore.
- Restarting HADR on the standby system.

However, if the problem is noticed with the standby database after a takeover has occurred (or if a choice was made to not address the storage issues until this time) then the resolution is based on the type of problem that was encountered.

If the database is enabled for automatic storage and space is not available on the storage paths associated with the standby database then follow these steps:

1. Make space available on the storage paths by extending the file systems, or by removing unnecessary non-DB2 files on them.
2. Perform a table space rollforward to the end of logs.

In the case where the addition or extension of containers as part of log replay could not occur, if the necessary backup images and log file archives are available, you might be able to recover the table space by first issuing the SET TABLESPACE CONTAINERS statement with the IGNORE ROLLFORWARD CONTAINER OPERATIONS option and then issuing the **ROLLFORWARD** command.

The primary and standby databases do not require the same database path. If relative container paths are used, the same relative path might map to different absolute container paths on the primary and standby databases. Consequently, if the primary and standby databases are placed on the same computer, all table space containers must be defined with relative paths so that they map to different paths for primary and standby.

HADR and Network Address Translation (NAT) support

NAT, which is supported in an HADR environment, is usually used for firewall and security because it hides the server's real address.

In an HADR setup, the local and remote host configurations on the primary and standby nodes are cross-checked to ensure they are correct. In a NAT environment, a host is known to itself by a particular IP address but is known to the other hosts by a different IP address. This behavior causes the HADR host cross-check to fail unless you set the **DB2_HADR_NO_IP_CHECK** registry variable to `ON`. Using this setting causes the host cross-check to be bypassed, enabling the primary and standby to connect in a NAT environment.

If you are not running in a NAT environment, use the default setting of `OFF` for the **DB2_HADR_NO_IP_CHECK** registry variable. Disabling the cross-check weakens the HADR validation of your configuration.

Considerations for HADR multiple standby mode

In a NAT environment with a multiple standby setup, each standby's settings for **hadr_local_host** and **hadr_local_svc** must still be listed in the primary's **hadr_target_list** or the primary does not accept the connection from that standby.

Normally, in multiple standby mode, on start up, a standby checks that its settings for **hadr_remote_host** and **hadr_remote_svc** are in its **hadr_target_list**, to ensure that on role switch, the old primary can become a new standby. In NAT scenarios, that check fails unless the **DB2_HADR_NO_IP_CHECK** registry variable to ON. Because this check is bypassed, the standby waits until it connects to the primary to check that the primary's **hadr_local_host** and **hadr_local_svc** are in the standby's **hadr_target_list**. The check still ensures role switch can succeed on this pair.

Note: If the **DB2_HADR_NO_IP_CHECK** registry variable is set to ON, the **hadr_remote_host** and **hadr_remote_svc** are not automatically updated.

In a multiple standby setup, **DB2_HADR_NO_IP_CHECK** should be set on all databases that might be making a connection to another database across a NAT boundary. If a database will never cross a NAT boundary to connect to another database (that is, if no such link is configured), then you should not set this registry variable on that database. When **DB2_HADR_NO_IP_CHECK** is set, it prevents a standby from automatically discovering the new primary after a takeover has occurred, and you have to manually reconfigure the standby to have it connect to the new primary.

Restrictions for High Availability Disaster Recovery (HADR)

To achieve optimal performance with High Availability Disaster Recovery (HADR), consider HADR restrictions when designing your high availability DB2 database solution.

The following list is a summary of High Availability Disaster Recovery (HADR) restrictions:

- HADR is not supported in a partitioned database environment.
- HADR is not supported in DB2 pureScale environments.
- The primary and standby databases must have the same operating system version and the same version of the DB2 database system, except for a short time during a rolling upgrade.
- The DB2 database system software on the primary and standby databases must be the same bit size (32 or 64 bit).
- Clients cannot connect to the standby database unless you have reads on standby enabled. Reads on standby enables clients to connect to the active standby database and issue read-only queries.
- If reads on standby is enabled, operations on the standby database that write a log record are not permitted; only read clients can connect to the active standby database.
- If reads on standby is enabled, write operations that would modify database contents are not allowed on the standby database. Any asynchronous threads such as real-time statistics collection, Auto Index rebuild and utilities that attempt to modify database objects will not be supported. Real-time statistics collection and Auto Index rebuild should not be running on the standby database.
- Log files are only archived by the primary database.
- The self tuning memory manager (STMM) can be run only on the current primary database. After the primary database is started or the standby database is converted to a primary database by takeover, the STMM EDU may not start until the first client connection comes in.
- Backup operations are not supported on the standby database.
- The **SET WRITE** command cannot be issued on the standby database

- Non-logged operations, such as changes to database configuration parameters and to the recovery history file, as well as LOB table columns that have the NOT LOGGED option, are not replicated to the standby database.
- Load operations with the **COPY** NO option specified are not supported.
- HADR does not support the use of raw I/O (direct disk access) for database log files. If HADR is started via the **START HADR** command, or the database is activated (restarted) with HADR configured, and raw logs are detected, the associated command will fail.
- Federated server does not fully support HADR in federated two phase commit (F2PC) scenarios. When a HADR database is configured as a federated database, it only supports F2PC with type-1 inbound connections.
- HADR does not support infinite logging.
- The system clock of the HADR primary database must be synchronized with the HADR standby database's system clock.

DB2 High availability disaster recovery (HADR) management

DB2 High availability disaster recovery (HADR) management involves configuring and maintaining the status of your HADR system.

Managing HADR includes such tasks as:
- Cataloging an HADR database.
- "Initializing high availability disaster recovery (HADR)" on page 894
- Checking or altering database configuration parameters related to HADR.
- "Switching database roles in high availability disaster recovery (HADR)" on page 921
- "Performing an HADR failover operation" on page 919
- "Monitoring high availability disaster recovery (HADR) environments" on page 923
- "Stopping DB2 High Availability Disaster Recovery (HADR)" on page 925

You can manage HADR using the following methods:
- Command line processor
- DB2 administrative API
- Task assistants for managing HADR in IBM Data Studio Version 3.1 or later.

Related information:

Administering databases with task assistants

DB2 High Availability Disaster Recovery (HADR) commands

The DB2 High Availability Disaster Recovery (HADR) feature provides complex logging, failover, and recovery functionality for DB2 high availability database solutions.

Despite the complexity of the functionality HADR provides, there are only a few actions you need to directly command HADR to perform: starting HADR; stopping HADR; and causing the standby database to take over as the primary database.

There are three high availability disaster recover (HADR) commands used to manage HADR:
- **START HADR**

- **STOP HADR**
- **TAKEOVER HADR**

To invoke these commands, use the command line processor or the administrative API.

Issuing the **START HADR** command with either the AS PRIMARY or AS STANDBY option changes the database role to the one specified if the database is not already in that role. This command also activates the database, if it is not already activated.

The **STOP HADR** command changes an HADR database (either primary or standby) into a standard database. Any database configuration parameters related to HADR remain unchanged so that the database can easily be reactivated as an HADR database.

The **TAKEOVER HADR** command, which you can issue on the standby database only, changes the standby database to a primary database. When you do not specify the BY FORCE option, the primary and standby databases switch roles. When you do specify the BY FORCE option, the standby database unilaterally switches to become the primary database. In this case, the standby database attempts to stop transaction processing on the old primary database. However, there is no guarantee that transaction processing will stop. Use the BY FORCE option to force a takeover operation for failover conditions only. To whatever extent possible, ensure that the current primary has definitely failed, or shut it down yourself, prior to issuing the **TAKEOVER HADR** command with the BY FORCE option.

HADR database role switching

A database can be switched between primary and standard roles dynamically and repeatedly. When the database is either online or offline, you can issue both the **START HADR** command with the AS PRIMARY option and the **STOP HADR** command.

You can switch a database between standby and standard roles statically. You can do so repeatedly only if the database remains in rollforward pending state. You can issue the **START HADR** command with the AS STANDBY option to change a standard database to standby while the database is offline and in rollforward pending state. Use the **STOP HADR** command to change a standby database to a standard database while the database is offline. The database remains in rollforward pending state after you issue the **STOP HADR** command. Issuing a subsequent **START HADR** command with the AS STANDBY option returns the database to standby. If you issue the **ROLLFORWARD DATABASE** command with the STOP option after stopping HADR on a standby database, you cannot bring it back to standby. Because the database is out of rollforward pending state, you can use it as a standard database. This is referred to as taking a snapshot of the standby database. After changing an existing standby database into a standard database, consider creating a new standby database for high availability purposes.

To switch the role of the primary and standby databases, perform a takeover operation without using the BY FORCE option.

To change the standby to primary unilaterally (without changing the primary to standby), use forced takeover. Subsequently, you might be able to reintegrate the old primary as a new standby.

HADR role is persistent. Once an HADR role is established, it remains with the database, even through repeated stopping and restarting of the DB2 instance or deactivation and activation of the DB2 database.

Starting the standby is asynchronous

When you issue the **START HADR** command with the AS STANDBY option, the command returns as soon as the relevant engine dispatchable units (EDUs) are successfully started. The command does not wait for the standby to connect to the primary database. In contrast, the primary database is not considered started until it connects to a standby database (with the exception of when the **START HADR** command is issued on the primary with the BY FORCE option). If the standby database encounters an error, such as the connection being rejected by the primary database, the **START HADR** command with the AS STANDBY option might have already returned successfully. As a result, there is no user prompt to which HADR can return an error indication. The HADR standby will write a message to the DB2 diagnostic log and shut itself down. You should monitor the status of the HADR standby to ensure that it successfully connects with the HADR primary.

Replay errors, which are errors that the standby encounters while replaying log records, can also bring down the standby database. These errors might occur, for example, when there is not enough memory to create a buffer pool, or if the path is not found while creating a table space. You should continuously monitor the status of the standby database.

Do not run HADR commands from a client using a database alias enabled for client reroute

When automatic client reroute is set up, the database server has a predefined alternate server so that client applications can switch between working with either the original database server or the alternative server with only minimal interruption of the work. In such an environment, when a client connects to the database via TCP, the actual connection can go to either the original database or to the alternate database. HADR commands are implemented to identify the target database through regular client connection logic. Consequently, if the target database has an alternative database defined, it is difficult to determine the database on which the command is actually operating. Although an SQL client does not need to know which database it is connecting to, HADR commands must be applied on a specific database. To accommodate this limitation, HADR commands should be issued locally on the server machine so that client reroute is bypassed (client reroute affects only TCP/IP connections).

HADR commands must be run on a server with a valid license

The **START HADR**, **STOP HADR**, and **TAKEOVER HADR** commands require that a valid HADR license has been installed on the server where the command is executed. If the license is not present, these commands will fail and return a command-specific error code (SQL1767N, SQL1769N, or SQL1770N, respectively) along with a reason code of 98. To correct the problem, either install a valid HADR license using **db2licm**, or install a version of the server that contains a valid HADR license as part of its distribution.

Initializing high availability disaster recovery (HADR)

Use this procedure to set up and initialize a DB2 high availability disaster recovery (HADR) primary database and one standby database.

About this task

HADR can be initialized through the command line processor (CLP), or by calling the db2HADRStart API.

Procedure

To use the CLP to initialize HADR on your system for the first time:

1. Determine the host name, host IP address, and the service name or port number for each of the HADR databases.

   If a host has multiple network interfaces, ensure that the HADR host name or IP address maps to the intended one. You need to allocate separate HADR ports in `/etc/services` for each protected database. These cannot be the same as the ports allocated to the instance. The host name can only map to one IP address.

   Note: The instance names for the primary and standby databases do not have to be the same.
2. Create the standby database by restoring a backup image or by initializing a split mirror, based on the existing database that is to be the primary.

   In the following example, the **BACKUP DATABASE** and **RESTORE DATABASE** commands are used to initialize a standby database. In this case, an NFS mounted file system is accessible at both sites.

   Issue the following command at the primary database:

   ```
   BACKUP DB dbname TO /nfs1/backups/db2/dbname
   ```

   If the database already exists on the standby instance, drop it first for a clean start. Files from the existing database can interfere with HADR operation. For example, left over log files can lead the standby onto a log chain not compatible with the primary. Issue the following command to drop the database:

   ```
   DROP DB dbname
   ```

   Issue the following command at the standby database:

   ```
   RESTORE DB dbname FROM /nfs1/backups/db2/dbname
   ```

   The following example illustrates how to use the **db2inidb** utility to initialize the standby database using a split mirror of the primary database. This procedure is an alternative to the backup and restore procedure illustrated previously.

   Issue the following command at the standby database:

   ```
   DB2INIDB dbname AS STANDBY
   ```

   Note:

   a. The database names for the primary and standby databases must be the same.
   b. Do not issue the **ROLLFORWARD DATABASE** command on the standby database after the restore operation or split mirror initialization. The results of using a rollforward operation might differ slightly from replaying the logs using HADR on the standby database. If the databases are not identical, attempts to start the standby will fail.
   c. When creating the standby database using the **RESTORE DATABASE** command, ensure that the standby remains in rollforward-pending or rollforward-in-progress mode. This means that you cannot issue the **ROLLFORWARD DATABASE** command with either the `COMPLETE` option or the `STOP`

option. An error will be returned if the **START HADR** command with the AS STANDBY option is attempted on the database after rollforward is stopped.

d. The following **RESTORE DATABASE** command options should be avoided when setting up the standby database: TABLESPACE, INTO, REDIRECT, and WITHOUT ROLLING FORWARD.

e. When setting up the standby database using the **db2inidb** utility, do not use the SNAPSHOT or MIRROR options. You can specify the RELOCATE USING option to change one or more of the following configuration attributes: instance name, log path, and database path. However, you must not change the database name or the table space container paths.

3. On each of the databases, set the **hadr_local_host**, **hadr_local_svc**, and **hadr_syncmode** configuration parameters:

```
"UPDATE DB CFG FOR dbname USING
     HADR_LOCAL_HOST   hostname
     HADR_LOCAL_SVC    servicename
     HADR_SYNCMODE     syncmode"
```

The configuration parameters in this step, step 4, and step 5 must be set after the standby databases has been created. If they are set prior to creating the standby database, the settings on the standby database will reflect what is set on the primary database.

Note: This is a generic HADR setup; for more advanced configuration options and settings, see the related links.

4. Optional: Set the **hadr_target_list** configuration parameter on the standby and the primary:

```
UPDATE DB CFG FOR dbname USING
     HADR_TARGET_LIST  principalhostname:principalservicename[3]
```

This is an optional, but recommended, step if you are only using one standby database. If you set the **hadr_target_list** parameter, you can add additional standby database dynamically. You can also take advantage of the autoconfiguration behavior and specify a different synchronization mode on the standby.

5. On each of the databases, set the **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters.

On the primary, set the parameters to the corresponding values on the standby by issuing the following command:

```
"UPDATE DB CFG FOR dbname USING
     HADR_REMOTE_HOST   principalhostname
     HADR_REMOTE_SVC    principalservicename
     HADR_REMOTE_INST   principalinstname"
```

On the standby, set the parameters to the corresponding values on the primary by issuing the following command:

```
"UPDATE DB CFG FOR dbname USING
     HADR_REMOTE_HOST   primaryhostname
     HADR_REMOTE_SVC    primaryservicename
     HADR_REMOTE_INST   primaryinstname"
```

If you have configured **hadr_target_list**, the values for these parameters are automatically given the proper values if they were set incorrectly. However, explicitly setting them to the correct values makes correct values available immediately. These values are helpful for the IBM Tivoli System Automation

[3]

for Multiplatforms (SA MP) software, which might require the **hadr_remote_host** value to construct the resource name.

6. Connect to the standby instance and start HADR on the standby database, as in the following example:

   ```
   START HADR ON DB dbname AS STANDBY
   ```

   Note: Usually, the standby database is started first. If you start the primary database first, this startup procedure will fail if the standby database is not started within the time period specified by the **hadr_timeout** database configuration parameter.
   After the standby starts, it enters *local catchup* state in which locally available log files are read and replayed. After it has replayed all local logs, it enters *remote catchup pending* state.
7. Connect to the primary instance and start HADR on the primary database, as in the following example:

   ```
   START HADR ON DB dbname AS PRIMARY
   ```

   After the primary starts, the standby enters *remote catchup* state in which receives log pages from the primary and replays them. After it has replayed all log files that are on the disk of the primary database machine, both databases enter *peer* state (unless you have chosen SUPERASYNC as the synchronization mode).

Initializing a standby database

One strategy for making a database solution highly available is maintaining a primary database to respond to user application requests, and a secondary or standby database that can take over database operations for the primary database if the primary database fails.

Initializing the standby database entails copying the primary database to the standby database.

Procedure

There are several ways to initialize the standby database. For example:

- Use disk mirroring to copy the primary database, and use DB2 database suspended I/O support to split the mirror to create the second database.
- Create a backup image of the primary database and recovery that image to the standby database.
- Use SQL replication to capture data from the primary database and apply that data to the standby database.

What to do next

After initializing the standby database, you must configure your database solution to synchronize the primary database and standby database so the standby database can take over for the primary database if the primary database fails.

Using a split mirror as a standby database

Use the following procedure to create a split mirror of a database for use as a standby database outside of a DB2 pureScale environment.

If a failure occurs on the primary database and it becomes inaccessible, you can use the standby database to take over for the primary database.

About this task

If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it during rollforward operations. However, once the database is brought out of the rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. While the standby database will initially use a different log chain from the primary database, the primary database could eventually use the same log chain value as the standby database. This could cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa , and can affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid recoverability issues.

Procedure

To use a split mirror as a standby database:

1. Connect to the primary database using the following command:

   ```
   db2 connect to db_name
   ```

2. Suspend the I/O write operations on the primary database using the following command:

   ```
   db2 set write suspend for database
   ```

   Note: While the database is in suspended state, you should not be running other utilities or tools. You should only be making a copy of the database. You can optionally use the FLUSH BUFFERPOOLS ALL statement before issuing **SET WRITE** SUSPEND to minimize the recovery time of the standby database.

3. Create one or multiple split mirrors from the primary database using appropriate operating system-level and storage-level commands.

   Note:

   - Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
   - If you specified the EXCLUDE LOGS with the **SET WRITE** command, do not include the log files in the copy.

4. Resume the I/O write operations on the primary database using the following command:

   ```
   db2 set write resume for database
   ```

5. Catalog the mirrored database on the secondary system.

   Note: By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the RELOCATE USING option of the **db2inidb** command to accomplish this.

6. Start the database instance on the secondary system using the following command:

   ```
   db2start
   ```

7. Initialize the mirrored database on the secondary system by placing it in rollforward pending state using the following command:

   ```
   db2inidb <database_alias> as standby
   ```

   If required, specify the RELOCATE USING option of the **db2inidb** command to relocate the standby database:

   ```
   db2inidb <database_alias> as standby relocate using relocatedbcfg.txt
   ```

   where the relocatedbcfg.txt file contains the information required to relocate the database.

   Note: You can take a full database backup using the split mirror if you have DMS table spaces (database managed space) or automatic storage table spaces. Taking a backup using the split mirror reduces the overhead of taking a backup on the production database. Such backups are considered to be online backups and will contain in-flight transactions, but you cannot include log files from the standby database. When such a backup is restored, you must rollforward to at least the end of the backup before you can issue a **ROLLFORWARD** command with the STOP option. Because the backup will not contain any log files, the log files from the primary database that were in use at the time the **SET WRITE** SUSPEND command was issued must be available or the rollforward operation will not be able to reach the end of the backup.

8. Make the archived log files from the primary database available to the standby database either by configuring the log archiving parameters on the standby database or by shipping logs to the standby database.
9. Rollforward the database to the end of the logs or to a point-in-time.
10. Continue retrieving log files and rollforwarding the database through the logs until you reach the end of the logs or the point-in-time required for the standby database.
11. Bring the standby database online by issuing the **ROLLFORWARD** command with the STOP option specified.

    Note:

    - The logs from the primary database cannot be applied to the mirrored database after it has been taken out of rollforward pending state.
    - If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it while rollforward is being performed. However, once the database is brought out of rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. Although the standby database will initially use a different log chain from the primary database, there is nothing to prevent the primary database from eventually using the same log chain value as the standby database. This may cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa. This could affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid these issues.

Using a split mirror as a standby database in a DB2 pureScale environment

Use the following procedure to create a split mirror of a database for use as a standby database in a DB2 pureScale environment. If a failure occurs on the primary database and it becomes inaccessible, you can use the standby database to take over for the primary database.

About this task

If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it during rollforward operations. However, once the database is brought out of the rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. While the standby database will initially use a different log chain from the primary database, the primary database could eventually use the same log chain value as the standby database. This could cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa , and can affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid recoverability issues.

Procedure

To use a split mirror as a standby database:

1. Connect to the primary database using the following command:
   ```
   db2 connect to <db_namd>
   ```
2. Configure the General Parallel File System (GPFS) on the secondary cluster by extracting and importing the primary cluster's settings. On the primary cluster, run the following GPFS command:
   ```
   mmfsctl <filesystem> syncFSconfig -n <remotenodefile>
   ```

   where <*remotenodefile*> is the list of hosts in the secondary cluster.
3. List the cluster manager domain using the following command:
   ```
   db2cluster -cm -list -domain
   ```
4. Stop the cluster manager on each host in the cluster using the following command:
   ```
   db2cluster -cm -stop -host <host> -force
   ```

   Note: The last host which you shut down must be the host from which you are issuing this command.
5. Stop the GPFS cluster on the secondary system using the following command:
   ```
   db2cluster -cfs -stop -all
   ```
6. Suspend the I/O write operations on the primary database using the following command:
   ```
   db2 set write suspend for database
   ```

   Note: While the database is in suspended state, you should not be running other utilities or tools. You should only be making a copy of the database. You can optionally flush all buffer pools before issuing **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved using the FLUSH BUFFERPOOLS ALL statement.

7. Determine which file systems must be suspended and copied using the following command:

```
db2cluster -cfs -list -filesystem
```

8. Suspend each GPFS file system that contains data or log data using the following command:

```
/usr/lpp/mmfs/bin/mmfsctl <filesystem> suspend
```

   where *<filesystem>* represents a file system that contains data or log data.

   Note: While the GPFS file systems are suspended, both read and write operations are blocked. You should only be performing the split mirror operations during this period to minimize the amount of time that read operations are blocked.

9. Create one or multiple split mirrors from the primary database using appropriate operating system-level and storage-level commands.

   Note:

   - Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
   - If you specified the `EXCLUDE LOGS` with the **`SET WRITE`** command, do not include the log files in the copy.

10. Resume the GPFS file systems that were suspended using the following command for each suspended file system:

```
/usr/lpp/mmfs/bin/mmfsctl <filesystem> resume
```

   where *filesystem* represents a suspended file system that contains data or log data.

11. Resume the I/O write operations on the primary database using the following command:

```
db2 set write resume for database
```

12. Start the GPFS cluster on the secondary system using the following command:

```
 db2cluster -cfs -start -all
```

13. Start the cluster manager using the following command

```
db2cluster -cm -start -domain <domain>
```

14. Catalog the mirrored database on the secondary system.

   Note: By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the **`RELOCATE USING`** option of the **`db2inidb`** command to accomplish this.

15. Start the database instance on the secondary system using the following command:

```
db2start
```

16. Initialize the database on the secondary system by placing it in rollforward pending state:

```
db2inidb <database_alias> as standby
```

If required, specify the RELOCATE USING option of the **db2inidb** command to relocate the database:

```
db2inidb database_alias as standby relocate using relocatedbcfg.txt
```

where relocatedbcfg.txt contains the information required to relocate the database.

Note: You can take a full database backup using the split mirror if you have DMS table spaces (database managed space) or automatic storage table spaces. Taking a backup using the split mirror reduces the overhead of taking a backup on the production database. Such backups are considered to be online backups and will contain in-flight transactions, but you cannot include log files from the standby database. When such a backup is restored, you must rollforward to at least the end of the backup before you can issue a **ROLLFORWARD** STOP command. Because the backup will not contain any log files, the log files from the primary database that were in use at the time the **SET WRITE** SUSPEND command was issued must be available or the rollforward operation will not be able to reach the end of the backup.

17. Make the archived log files from the primary database available to the standby database either by configuring the log archiving parameters on the standby database or by shipping logs to the standby database.
18. Rollforward the database to the end of the logs or to a point-in-time.

    Note: When executing rollforward operations, you might encounter SQL1273 errors. These errors are expected if some of the log files were not copied from the primary system when the database was split or if one member generates log files faster than other members. SQL1273 is generated in some cases when the rollforward operation must stop to preserve data consistency because the contents of the log files depends on the contents of unavailable log files from other members. If the standby database is configured to retrieve log files archived by the primary database, you can either wait for the primary system to archive the necessary log file or you can use the **ARCHIVE LOG** command on the primary system to force the log file to be archived. Otherwise, you must ship the required log files to the standby database. After the necessary log file is available on the standby database, the rollforward operation can read further ahead in the logs, although SQL1273 might be encountered again if some members are still generating log files faster than other members. For more information, see the “Disaster recovery and high availability through log shipping in a DB2 pureScale environment” section of the “Backup and restore operations in a DB2 pureScale environment” Information Center topic.
19. Continue the rollforward operation through the logs until you reach the end of the logs or the point-in-time required for the standby database, shipping new log files to the standby database if required.
20. Bring the standby database online by issuing the **ROLLFORWARD DATABASE** command with the STOP option specified.

    Note:

    - The logs from the primary database cannot be applied to the mirrored database once it has been taken out of rollforward pending state.
    - If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it while rollforward is being performed. However, once the database is brought out of rollforward

pending state, the standby database will attempt to archive log files to the same location used by the primary database. Although the standby database will initially use a different log chain from the primary database, there is nothing to prevent the primary database from eventually using the same log chain value as the standby database. This may cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa. This could affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid these issues.

Database configuration for high availability disaster recovery (HADR)

You can use database configuration parameters to help achieve optimal performance with DB2 HADR.

In most cases, you should use the same database configuration parameter settings and database manager configuration parameter settings on the systems where the primary and standby databases are located. If the settings for the configuration parameters on the standby database are different from the settings on the primary, the following problems might occur:

- Error messages might be returned for the standby database while the log files that were shipped from the primary database are being replayed.
- After a takeover operation, the new primary database might be unable to handle the workload, resulting in performance problems or in applications receiving error messages that they did not receive when they were connected to the original primary database.

Changes to the configuration parameters on the primary database are not automatically propagated to the standby database. You must manually make changes on the standby database. For dynamic configuration parameters, changes take effect without the need to shut down and restart the database management system (DBMS) or the database. For non-dynamic configuration parameters, changes take effect after the standby database is restarted.

Following are sections on specific configuration topics for HADR:

- “Size of log files configuration parameter on the standby database”
- “Database configuration parameter autorestart” on page 904
- “Log receive buffer size on a standby database” on page 904
- “Load operations and HADR” on page 904
- “DB2_HADR_PEER_WAIT_LIMIT registry variable” on page 906
- “HADR configuration parameters” on page 907

Size of log files configuration parameter on the standby database

One exception to the configuration parameter behavior that is described in the previous paragraph is the behavior of the **logfilsiz** database configuration parameter. Although the value of this parameter is not replicated to the standby database, to guarantee identical log files on both databases, the setting for the **logfilsiz** configuration parameter on the standby is ignored. Instead, the database creates local log files whose sizes match the size of the log files on the primary database.

After a takeover, the original standby (new primary) uses the **logfilsiz** parameter value that you set on the original primary until you restart the database. At that point, the new primary reverts to using the value that you set locally. In addition, the current log file is truncated and any pre-created log files are resized on the new primary.

If the databases keep switching roles as a result of a non-forced takeover and neither database is deactivated, the log file size that is used is always the one from the original primary database. However, if there is a deactivation and then a restart on the original standby (new primary), the new primary uses the log file size that you configured locally. This log file size continues to be used if the original primary takes over again. Only after a deactivation and restart on the original primary would the log file size revert to the settings on the original primary.

Database configuration parameter `autorestart`

The recommended configuration for the **autorestart** parameter on HADR systems is `ON`. If the **autorestart** parameter is set to `OFF`, and the server fails, your response depends on whether or not you want to restart or fail over to the standby:

- If you want to restart, run the **RESTART DATABASE** command manually. If the restart fails, perform failover.
- If you want to fail over, perform the following steps:
  1. Shut down the old primary to prevent a “split brain”. Do this by either stopping the DB2 instance or powering off the host machine. If the server is not accessible for administration, fence it off from clients by disabling the client/server network.

     Note: Deactivating the database is not sufficient because client connections can bring it back online. If it failed in a consistent state, then even if the **autorestart** parameter is set to `OFF`, this does not prevent client connections from bringing it back online.
  2. After shutting down old primary, issue the **TAKEOVER HADR** command with the `BY FORCE` option on the standby.

Log receive buffer size on a standby database

By default, the log receive buffer size on a standby database is two times the value that you specify for the **logbufsz** configuration parameter on the primary database. This size might not be sufficient. For example, consider what might happen when the HADR synchronization mode is set to ASYNC and the primary and standby databases are in peer state. If the primary database is also experiencing a high transaction load, the log receive buffer on the standby database might fill to capacity, and the log shipping operation from the primary database might stall. To manage these temporary peaks, you can make either of the following configuration changes:

- Increase the size of the log receive buffer on the standby database by modifying the value of the **DB2_HADR_BUF_SIZE** registry variable.
- Enable log spooling on a standby database by setting the **hadr_spool_limit** configuration parameter.

Load operations and HADR

If you issue the **LOAD** command on the primary database with the **COPY YES** parameter, the command executes on the primary database, and the data is replicated to the standby database if the load copy can be accessed through the path or device that is specified by the command. If load copy data cannot be accessed from the standby database, the table space in which the table is stored is marked invalid on the standby database. Any future log records that pertain to this table space are skipped. To ensure that the load operation can access the load copy on the standby database, use a shared location for the output file from the **COPY YES** parameter. Alternatively, you can deactivate the standby database while performing the load on the primary, place a copy of the output file in the standby path, and then activate the standby database.

If you issue the **LOAD** command with the **NONRECOVERABLE** parameter on the primary database, the command executes on the primary database, and the table on the standby database is marked invalid. Any future log records that pertain to this table are skipped. You can issue the **LOAD** command with the **COPY YES** and **REPLACE** parameters to bring the table back, or you can drop the table to recover the space.

Note: You cannot bring a table back using the LOAD command with the COPY YES and REPLACE options if the table has one of the following characteristics:
- The table was created with the NOT LOGGED INITIALLY attribute.
- The table is a multidimensional clustered (MDC) table.
- The table has compression dictionaries.
- The table has XML columns.

Because a load operation with the **COPY NO** parameter is not supported with HADR, the operation is automatically converted to a load operation with the **NONRECOVERABLE** parameter. To enable a load operation with the **COPY NO** parameter to be converted to a load operation with the **COPY YES** parameter, set the **DB2_LOAD_COPY_NO_OVERRIDE** registry variable on the primary database. This registry variable is ignored on the standby database. Ensure that the device or directory that you specify for the primary database can be accessed by the standby database by using the same path, device, or load library.

If you are using the Tivoli Storage Manager (TSM) software to perform a load operation with the **COPY YES** parameter, you might have to set the **vendoropt** configuration parameter on the primary and standby databases. Depending on how you configured TSM, the values on the primary and standby databases might not be the same. Also, when using TSM to perform a load operation with the **COPY YES** parameter, you must issue the **db2adutl** command with the **GRANT** parameter to give the standby database read access to the files that are loaded.

If table data is replicated by a load operation with the **COPY YES** parameter, the indexes are replicated as follows:
- If you specify the REBUILD indexing mode option with the **LOAD** command and the LOG INDEX BUILD table attribute is set to ON (using the ALTER TABLE statement), or if it is set to NULL and the **logindexbuild** database configuration parameter is set to ON, the primary database includes the rebuilt index object (that is, all of the indexes defined on the table) in the copy file to enable the standby database to replicate the index object. If the index object on the standby database is marked invalid before the load operation, it becomes usable again after the load operation as a result of the index rebuild.
- If you specify the INCREMENTAL indexing mode option with the **LOAD** command and the LOG INDEX BUILD table attribute is set to ON (using the ALTER TABLE statement), or if it is set to NULL and the **logindexbuild** database configuration parameter on the primary database is set to ON, the index object on the standby

database is updated only if it is not marked invalid before the load operation. Otherwise, the index is marked invalid on the standby database.

DB2_HADR_PEER_WAIT_LIMIT registry variable

Restriction: In multiple standby mode, none of this section applies to the auxiliary standbys because they are in SUPERASYNC synchronization mode, so they do not ever enter peer state.

If you set the **`DB2_HADR_PEER_WAIT_LIMIT`** registry variable, the HADR primary database breaks out of peer state if logging on the primary database has been blocked for the specified number of seconds because of log replication to the standby. When this limit is reached, the primary database breaks the connection to the standby database. If you disable the peer window by setting the **`hadr_peer_window`** configuration parameter to `0`, the primary enters the disconnected state, and logging resumes. If you enable the peer window, the primary database enters disconnected peer state, in which logging continues to be blocked. The primary leaves disconnected peer state upon reconnection or peer window expiration. Logging resumes after the primary leaves disconnected peer state.

Note: If you set **`DB2_HADR_PEER_WAIT_LIMIT`**, use a minimum value of `10` to avoid triggering false alarms.

Honoring peer window transition when a database breaks out of peer state ensures peer window semantics for safe takeover in all cases. If the primary fails during the transition, normal peer window protection still applies: safe takeover from the standby if it is still in disconnected peer state.

On the standby side, after disconnection, the database continues replaying already received logs. After the received logs have been replayed, the standby reconnects to the primary. After replaying the received logs, the standby reconnects to the primary. Upon reconnection, normal state transition follows: first remote catchup state, then peer state.

Relationship to hadr_timeout database configuration parameter

The **`hadr_timeout`** database configuration parameter does not break the primary out of peer state if the primary keeps receiving heartbeat messages from the standby while blocked. The **`hadr_timeout`** database configuration parameter specifies a timeout value for the HADR network layer. An HADR database breaks the connection to its partner database if it has not received any message from its partner for the period that is specified by the **`hadr_timeout`** configuration parameter. The timeout does not control timeout for higher-layer operations such as log shipping and ack (acknowledgement) signals. If log replay on the standby database is stuck on a large operation such as load or reorganization, the HADR component still sends heartbeat messages to the primary database on the normal schedule. In such a scenario, the primary is blocked as long as the standby replay is blocked unless you set the **`DB2_HADR_PEER_WAIT_LIMIT`** registry variable.

The **`DB2_HADR_PEER_WAIT_LIMIT`** registry variable unblocks primary logging regardless of connection status. Even if you do not set the **`DB2_HADR_PEER_WAIT_LIMIT`** registry variable, the primary always breaks out of peer state when a network error is detected or the connection is closed, possibly as result of the **`hadr_timeout`** configuration parameter.

HADR configuration parameters

Some HADR configuration parameters are static, such as **hadr_local_host** and **hadr_remote_host** . Static parameters are loaded on database startup, and changes are ignored during run time. HADR parameters are also loaded when the **START HADR** command completes. On the primary database, HADR can be started and stopped dynamically, with the database remaining online. Thus, one way to refresh the effective value of an HADR configuration parameter without shutting down the database is to stop and restart HADR. In contrast, the **STOP HADR** brings down the database on the standby, so the standby's parameters cannot be refreshed with database online.

Host name parameters and service and port name parameters (single standby mode) An HADR pair has five interrelated configuration parameters that you should set:

- **hadr_local_host**
- **hadr_remote_host**
- **hadr_local_svc**
- **hadr_remote_svc**
- **hadr_remote_inst**

TCP connections are used for communication between the primary and standby databases. The “local” parameters specify the local address and the “remote” parameters specify the remote address. A primary database listens on its local address for new connections. A standby database that is not connected to a primary database retries connection to its remote address.

The standby database also listens on its local address. In some scenarios, another HADR database can contact the standby database on this address and send it messages.

Unless the **HADR_NO_IP_CHECK** registry variable is set, HADR does the following cross-checks of local and remote addresses on connection:

```
 my local address = your remote address
```

and

```
my remote address = your local address
```

The check is done using the IP address and port number, rather than the literal string in the configuration parameters. You need to set the **HADR_NO_IP_CHECK** registry variable in NAT (Network Address Translation) environment to bypass the check.

You can configure an HADR database to use either IPv4 or IPv6 to locate its partner database. If the host server does not support IPv6, you must use IPv4. If the server supports IPv6, whether the database uses IPv4 or IPv6 depends upon the format of the address that you specify for the **hadr_local_host** and **hadr_remote_host** configuration parameters. The database attempts to resolve the two parameters to the same IP format and use IPv6 when possible. The following table shows how the IP mode is determined for IPv6-enabled servers:

IP mode used for **hadr_local_host** parameter	IP mode used for **hadr_remote_host** parameter	IP mode used for HADR communications
IPv4 address	IPv4 address	IPv4

IP mode used for `hadr_local_host` parameter	IP mode used for `hadr_remote_host` parameter	IP mode used for HADR communications
IPv4 address	IPv6 address	Error
IPv4 address	host name, maps to IPv4 only	IPv4
IPv4 address	host name, maps to IPv6 only	Error
IPv4 address	host name, maps to IPv4 and v6	IPv4
IPv6 address	IPv4 address	Error
IPv6 address	IPv6 address	IPv6
IPv6 address	host name, maps to IPv4 only	Error
IPv6 address	host name, maps to IPv6 only	IPv6
IPv6 address	host name, maps to IPv4 and IPv6	IPv6
hostname, maps to IPv4 only	IPv4 address	IPv4
hostname, maps to IPv4 only	IPv6 address	Error
hostname, maps to IPv4 only	hostname, maps to IPv4 only	IPv4
hostname, maps to IPv4 only	hostname, maps to IPv6 only	Error
hostname, maps to IPv4 only	hostname, maps to IPv4 and IPv6	IPv4
hostname, maps to IPv6 only	IPv4 address	Error
hostname, maps to IPv6 only	IPv6 address	IPv6
hostname, maps to IPv6 only	hostname, maps to IPv4 only	Error
hostname, maps to IPv6 only	hostname, maps to IPv6 only	IPv6
hostname, maps to IPv6 only	hostname, maps to IPv4 and IPv6	IPv6
hostname, maps to IPv4 and IPv6	IPv4 address	IPv4
hostname, maps to IPv4 and IPv6	IPv6 address	IPv6
hostname, maps to IPv4 and IPv6	hostname, maps to IPv4 only	IPv4
hostname, maps to IPv4 and IPv6	hostname, maps to IPv6 only	IPv6
hostname, maps to IPv4 and IPv6	hostname, maps to IPv4 and IPv6	IPv6

The primary and standby databases can make HADR connections only if they use the same IPv4 or IPv6 format. If one server is IPv6 enabled (but also supports IPv4) and the other server supports IPv4 only, at least one of the h**hadr_local_host** and **hadr_remote_host** parameters on the IPv6 server must specify an IPv4 address to force database on this server to use IPv4.

You can set the HADR local service and remote service parameters (**hadr_local_svc** and **hadr_remote_svc**) to either a port number or a service name. The values that you specify must map to ports that are not being used by any other service, including other DB2 components or other HADR databases. In particular, you cannot set either parameter value to the TCP/IP port that is used by the server to await communications from remote clients (the value of the **svcename** database manager configuration parameter) or the next port (the value of the **svcename** parameter + 1).

If the primary and standby databases are on different servers, they can use the same port number or service name; otherwise, they must have different values.

Host name, service or port name, and target list parameters (multiple standby mode)

In multiple standby mode, you should still set the **hadr_local_host**, **hadr_local_svc**, **hadr_remote_host**, **hadr_remote_host**, and **hadr_remote_inst** configuration parameters. If you set those parameters incorrectly, they are automatically updated after the primary connects to the standbys by using the settings of the **hadr_target_list** configuration parameter. This parameter specifies the host and port names of all the standbys. The first standby that you specify in the target list is considered to be the *principal HADR standby database*.

In multiple standby mode, you should still set the **hadr_local_host**, **hadr_local_svc**, **hadr_remote_host**, **hadr_remote_host**, and **hadr_remote_inst** configuration parameters. The **hadr_local_host** and **hadr_local_svc** parameters have the same meaning as in single standby mode. On the primary, set**hadr_remote_host**, **hadr_remote_host**, and **hadr_remote_inst** to indicate its principal standby. A new parameter, **hadr_target_list** is used to list all standbys, with the first entry being the principal standby. On standby, set the "remote" parameters to indicate the primary. In certain conditions, the "remote" parameters (on both the primary and the standby) can be automatically updated. For more information, see the "Automatic reconfiguration of HADR parameters" section in "Database configuration for multiple HADR standby databases" on page 859.

Synchronization mode

In single standby mode, the synchronization mode, which you specify with the **hadr_syncmode** configuration parameter must be identical on the primary and standby databases. The consistency of the value of this configuration parameter is checked when an HADR pair establishes a connection.

In multiple standby mode, the synchronization mode does not have to be the same. All standbys have an *effective synchronization mode* that is determined by the type of standby that they are. The principal standby uses the synchronization mode of the primary, and the auxiliary standbys use SUPERASYNC. All standbys have a *configured synchronization mode*, which is the explicit setting for **hadr_syncmode** and is used if a standby becomes the new primary.

For more detailed information, see "DB2 high availability disaster recovery (HADR) synchronization mode".

HADR timeout and peer window

The timeout period, which you specify with the **hadr_timeout** configuration parameter, must be identical on the primary and standby databases. The consistency of the values of these configuration parameters is checked when an HADR pair establishes a connection.

With one exception, when the primary database starts, it waits for the longer of the two following periods for a standby to connect:

- For a minimum of 30 seconds
- For the number of seconds that is specified by the **hadr_timeout** database configuration parameter.

If the standby does not connect in the specified time, the startup fails. The one exception to this behavior is when you issue the **START HADR** command with the **BY FORCE** parameter. In this case, the primary database starts without waiting for the standby database to connect to it.

In multiple standby mode, the primary only waits for the principal standby to connect; a connection to an auxiliary standby is optional.

After an HADR pair establishes a connection, they exchange heartbeat messages. The heartbeat interval is computed from factors like the **hadr_timeout** and **hadr_peer_window** configuration parameters. It is reported by the HEARTBEAT_INTERVAL field in MON_GET_HADR table function. If one database does not receive any message from the other database within the number of seconds that is specified by the **hadr_timeout** configuration parameter, it initiates a disconnect. This behavior means that at most, it takes the number of seconds that is specified by the **hadr_timeout** configuration parameter for an HADR database to detect the failure of either its partner database or the intervening network. If you set the **hadr_timeout** configuration parameter too low, you will receive false alarms and frequent disconnections.

If you have the **hadr_peer_window** configuration parameter set to a nonzero value and the primary loses connection to the standby in peer state, the primary database does not commit transactions until the connection with the standby database is restored or the time value of the **hadr_peer_window** configuration parameter elapses, whichever happens first.

For maximal availability, the default value for the **hadr_peer_window** database configuration parameter is 0. When this parameter is set to 0, as soon as the connection between the primary and the standby is closed, the primary drops out of peer state to avoid blocking transactions. The connection can close because the standby closed the connection, a network error is detected, or timeout is reached. For increased data consistency, but reduced availability, you can set the **hadr_peer_window** database configuration parameter to a nonzero value.

For more information, see “Setting the **hadr_timeout** and **hadr_peer_window** database configuration parameters”.

The following sample configuration is for the primary and standby databases:

Primary database:

```
HADR_LOCAL_HOST          host1.ibm.com
HADR_LOCAL_SVC           hadr_service
HADR_REMOTE_HOST         host2.ibm.com
HADR_REMOTE_SVC          hadr_service
HADR_REMOTE_INST         dbinst2
HADR_TIMEOUT             120
HADR_SYNCMODE            NEARSYNC
HADR_PEER_WINDOW         120
```

Standby database:

```
HADR_LOCAL_HOST          host2.ibm.com
HADR_LOCAL_SVC           hadr_service
HADR_REMOTE_HOST         host1.ibm.com
HADR_REMOTE_SVC          hadr_service
HADR_REMOTE_INST         dbinst1
HADR_TIMEOUT             120
HADR_SYNCMODE            NEARSYNC
HADR_PEER_WINDOW         120
```

Setting the hadr_timeout and hadr_peer_window database configuration parameters

You can configure the **hadr_timeout** and **hadr_peer_window** database configuration parameters for optimal response to a connection failure.

hadr_timeout database configuration parameter

If an HADR database does not receive any communication from its partner database for longer than the length of time specified by the **hadr_timeout** database configuration parameter, then the database concludes that the connection with the partner database is lost. If the database is in peer state when the connection is lost, then it moves into disconnected peer state if the **hadr_peer_window** database configuration parameter is greater than zero, or into remote catchup pending state if **hadr_peer_window** is not greater than zero. The state change applies to both primary and standby databases.

hadr_peer_window database configuration parameter

The **hadr_peer_window** configuration parameter does not replace the **hadr_timeout** configuration parameter. The **hadr_timeout** configuration parameter determines how long an HADR database waits before considering its connection with the partner database as failed. The **hadr_peer_window** configuration parameter determines whether the database goes into disconnected peer state after the connection is lost, and how long the database should remain in that state. HADR breaks the connection as soon as a network error is detected during send, receive, or poll on the TCP socket. HADR polls the socket every 100 milliseconds. This allows it to respond quickly to network errors detected by the OS. Only in the worst case does HADR wait until the timeout to break a bad connection. In this case, a database application that is running at the time of failure can be blocked for a period of time equal to the sum of the **hadr_timeout** and **hadr_peer_window** database configuration parameters.

Setting the hadr_timeout and hadr_peer_window database configuration parameters

It is desirable to keep the waiting time that a database application experiences to a minimum. Setting the **hadr_timeout** and **hadr_peer_window** configuration parameters to small values would reduce the time that a database application must wait if a HADR standby databases loses its connection with the primary database. However, there are two other details that should be considered when choosing values to assign to the **hadr_timeout** and **hadr_peer_window** configuration parameters:

- The **hadr_timeout** database configuration parameter should be set to a value that is long enough to avoid false alarms on the HADR connection caused by short, temporary network interruptions. For example, the default value of **hadr_timeout** is 120 seconds, which is a reasonable value on many networks.
- The **hadr_peer_window** database configuration parameter should be set to a value that is long enough to allow the system to perform automated

failure responses. If the HA system, for example a cluster manager, detects primary database failure before disconnected peer state ends, a failover to the standby database takes place. Data is not lost in the failover as all data from old primary is replicated to the new primary. If **hadr_peer_window** is too short, HA system may not have enough time to detect the failure and respond.

Note: In HADR multiple standby mode, the principal standby uses the primary's setting for **hadr_peer_window** (the *effective peer window*). The setting for **hadr_peer_window** on any auxiliary standby is meaningless because that type of standby always runs in SUPERASYNC mode.

Log archiving configuration for DB2 high availability disaster recovery (HADR)

To use log archiving with DB2 high availability disaster recovery (HADR), configure both the primary database and the standby database for automatic log retrieval capability from all log archive locations. For multiple standby systems, configure archiving on primary and all standby databases.

Only the current primary database can perform log archiving. If the primary and standby databases are set up with separate archiving locations, logs are archived only to the primary database's archiving location. In the event of a takeover, the standby database becomes the new primary database and any logs archived from that point on are saved to the original standby database's archiving location. In such a configuration, logs are archived to one location or the other, but not both; with the exception that following a takeover, the new primary database might archive a few logs that the original primary database had already archived. In a multiple standby system, the archived log files can be scattered among all databases' (primary and standbys) archive devices. A shared archive is preferred because all files are stored in a single location.

Many operations need to retrieve archived log files. These operations include: database roll forward, the HADR primary database retrieving log files to send to the standby database in remote catch up, and replication programs (such as Q Replication) reading logs. As a result, a shared archive for an HADR system is preferred, otherwise, the needed files can be distributed on multiple archive devices, and user intervention is needed to locate the needed files and copy them to the requesting database. The recommended copy destination is an archive device. If copying into an archive is not feasible, copy the logs into the overflow log path. As a last resort, copy them into the log path (but you should be aware that there is a risk of damaging the active log files). DB2 does not auto delete user copied files in the overflow and active log path, so you should manually remove the files when they are no longer needed by any HADR standby or any application.

A specific scenario is a takeover in multiple standby mode. After the takeover, the new primary might not have all log files needed by other standbys (because a standby is at an older log position). If the primary cannot find a requested log file, it notifies the standby, which closes the connection and then reconnects in a few seconds to retry. The retry duration is limited to a few minutes. When retry time is exhausted, the standby shuts down. In this case, you should copy the files to the primary to ensure it has files from the first missing file to its current log file. After the files are copied, restart the standby if needed.

The standby database automatically manages log files in its log path. The standby database does not delete a log file from its local log path until it has been notified by the primary database that the primary database has archived it. This behavior provides added protection against the loss of log files. If the primary database fails and its log disk becomes corrupted before a particular log file is archived on the primary database, the standby database does not delete that log file from its own disk because it has not received notification that the primary database successfully archived the log file. If the standby database then takes over as the new primary database, it archives that log file before recycling it. If both the **logarchmeth1** and **logarchmeth2** configuration parameters are in use, the standby database does not recycle a log file until the primary database has archived it using both methods.

In addition to the benefits previously listed, a shared log archive device improves the catchup process by allowing the standby database to directly retrieve older log files from the archive in local catchup state, instead of retrieving those files indirectly through the primary in remote catchup state. However, it is recommended that you not use a serial archive device such as a tape drive for HADR databases. With serial devices, you might experience performance degradation on both the primary and standby databases because of mixed read and write operations. The primary writes to the device when it archives log files and the standby reads from the device to replay logs. This performance impact can occur even if the device is not configured as shared.

Shared log archives on Tivoli Storage Manager

Using a shared log archive with IBM Tivoli Storage Manager (TSM) allows one or more nodes to appear as a single node to the TSM server, which is especially useful in an HADR environment where either machine can be the primary at any one time.

To set up a shared log archive, you need to use proxy nodes which allow the TSM client nodes to perform data protection operations against a centralized name space on the TSM server. The target client node owns the data and agent nodes act on behalf of the target nodes to manage the backup data. The proxy node target is the node name defined on the TSM server to which backup versions of distributed data are associated. The data is managed in a single namespace on the TSM server as if it is entirely the data for this node. The proxy node target name can be a real node (for example, one of the application hosts) or a virtual node name (that is, with no corresponding physical node). To create a virtual proxy node name, use the following commands on the TSM server:

```
Grant proxynode target=virtual-node-name agent=HADR-primary-name
Grant proxynode target=virtual-node-name agent=HADR-standby-name
```

Next, you need to set these database configuration parameters on the primary and standby databases to the *virtual-node-name*:

- **vendoropt**
- **logarchopt**

In a multiple standby setup, you need to grade proxynode access to all machines on the TSM server and configure the **vendoropt** and **logarchopt** configuration parameters on all of the standbys.

HADR log spooling

The high availability disaster recovery (HADR) log spooling feature allows transactions on primary to make progress without having to wait for the log replay on the standby.

When this feature is enabled, log data sent by the primary is *spooled*, or written, to disk on the standby, and that log data is later read by log replay.

Log spooling, which is enabled by setting the **hadr_spool_limit** database configuration parameter, is an improvement to the HADR feature. When replay is slow, it is possible that new transactions on the primary can be blocked because it is not able to send log data to the standby system if there is no room in the buffer to receive the data. The log spooling feature means that the standby is not limited by the size of its buffer. When there is an increase in data received that cannot be contained in the buffer, the log replay reads the data from disk. This allows the system to better tolerate either a spike in transaction volume on the primary, or a slow down of log replay (due to the replay of particular type of log records) on the standby.

This feature could potentially lead to a larger gap between the log position of received logs on the standby and the log replay position on the standby, which can lead to longer takeover time. Use the **db2pd** command with the **-hadr** option or the MON_GET_HADR table function to monitor this gap by comparing the STANDBY_LOG_POS field, which shows receive position, and the STANDBY_REPLAY_LOG_POS field. You should consider your spool limit setting carefully because the old standby cannot start up as the new primary and receive transactions until the replay of the spooled logs has finished.

Index logging and high availability disaster recovery (HADR)

You should consider setting the database configuration parameters **logindexbuild** and **indexrec** for high availability disaster recovery (HADR) databases.

Using the logindexbuild database configuration parameter

Recommendation: For HADR databases, set the **logindexbuild** database configuration parameter to ON to ensure that complete information is logged for index creation, re-creation, and reorganization. Although this means that index builds might take longer on the primary system and that more log space is required, the indexes will be rebuilt on the standby system during HADR log replay and will be available when a failover takes place. Otherwise, when replaying an index build or rebuild event, the standby marks the index invalid, because the log records do not contain enough information to populate the new index. If index builds on the primary system are not logged and a failover occurs, any invalid indexes that remain after the failover is complete have to be rebuilt before they can be accessed. While the indexes are being re-created, they cannot be accessed by any applications.

Note: If the LOG INDEX BUILD table attribute is set to its default value of NULL, DB2 uses the value specified for the **logindexbuild** database configuration parameter. If the LOG INDEX BUILD table attribute is set to ON or OFF, the value specified for the **logindexbuild** database configuration parameter is ignored.

You might choose to set the LOG INDEX BUILD table attribute to OFF on one or more tables for either of the following reasons:

- You do not have enough active log space to support logging of the index builds.

- The index data is very large and the table is not accessed often; therefore, it is acceptable for the indexes to be re-created at the end of the takeover operation. In this case, set the **indexrec** configuration parameter to RESTART. Because the table is not frequently accessed, this setting causes the system to re-create the indexes at the end of the takeover operation instead of waiting for the first time the table is accessed after the takeover operation.

If the LOG INDEX BUILD table attribute is set to OFF on one or more tables, any index build operation on those tables might cause the indexes to be re-created any time a takeover operation occurs. Similarly, if the LOG INDEX BUILD table attribute is set to its default value of NULL, and the **logindexbuild** database configuration parameter is set to OFF, any index build operation on a table might cause the indexes on that table to be re-created any time a takeover operation occurs. You can prevent the indexes from being re-created by taking one of the following actions:

- After all invalid indexes are re-created on the new primary database, take a backup of the database and apply it to the standby database. As a result of doing this, the standby database does not have to apply the logs used for re-creating invalid indexes on the primary database, which would mark those indexes as rebuild required on the standby database.
- Set the LOG INDEX BUILD table attribute to ON, or set the LOG INDEX BUILD table attribute to NULL and the **logindexbuild** configuration parameter to ON on the standby database to ensure that the index re-creation will be logged.

Using the indexrec database configuration parameter

Recommendation: Set the **indexrec** database configuration parameter to RESTART (the default) on both the primary and standby databases. This causes invalid indexes to be rebuilt after a takeover operation is complete. If any index builds have not been logged, this setting allows DB2 to check for invalid indexes and to rebuild them. This process takes place in the background, and the database is accessible after the takeover operation has completed successfully.

If a transaction accesses a table that has invalid indexes before the indexes have been rebuilt by the background re-create index process, the invalid indexes are rebuilt by the first transaction that accesses it.

High availability disaster recovery (HADR) performance

Configuring different aspects of your database system, including network bandwidth, CPU power, and buffer size, can improve the performance of your DB2 high availability disaster recovery (HADR) databases.

The network is the key part of your HADR setup because network connectivity is required to replicate database changes from the primary to the standby, keeping the two databases in sync.

Recommendations for maximizing network performance:

- Ensure that network bandwidth is greater than the database log generation rate.
- Consider network delays when you choose the HADR synchronization mode. Network delays affect the primary only in SYNC and NEARSYNC modes.

  The slowdown in system performance as a result of using SYNC mode can be significantly larger than that of the other synchronization modes. In SYNC mode, the primary database sends log pages to the standby database only after the log pages are successfully written to the primary database log disk. To protect the integrity of the system, the primary database waits for an acknowledgment from the standby before it notifies an application that a transaction was prepared or committed. The standby database sends the acknowledgment only after it writes the received log pages to the standby database disk. The performance overhead equals the time that is needed for writing the log pages on the standby database plus the time that is needed for sending the messages back to the primary.

In NEARSYNC mode, the primary database writes and sends log pages in parallel. The primary then waits for an acknowledgment from the standby. The standby database acknowledges as soon as the log pages are received into its memory. On a fast network, the overhead to the primary database is minimal. The acknowledgment might have already arrived by the time the primary database finishes local log write.

For ASYNC mode, the log write and send are also in parallel; however, in this mode the primary database does not wait for an acknowledgment from the standby. Therefore, network delay is not an issue. Performance overhead is even smaller with ASYNC mode than with NEARSYNC mode.

For SUPERASYNC mode, transactions are never blocked or experience elongated response times because of network interruptions or congestion. New transactions can be processed as soon as previously submitted transactions are written to the primary database. Therefore, network delay is not an issue. The elapsed time for the completion of non-forced takeover operations might be longer than in other modes because the log gap between the primary and the standby databases might be relatively larger.

- Consider tuning the **DB2_HADR_SOSNDBUF** and **DB2_HADR_SORCVBUF** registry variables.

  HADR log shipping workload, network bandwidth, and transmission delay are important factors to consider when you are tuning the TCP socket buffer sizes. Two registry variables, **DB2_HADR_SOSNDBUF** and **DB2_HADR_SORCVBUF** allow tuning of the TCP socket send and receive buffer size for HADR connections only. These two variables have the value range of `1024` to `4294967295` and default to the socket buffer size of the operating system, which varies depending on the operating system. It is strongly recommended that you use a minimum value of `16384` (16 K) for your **DB2_HADR_SOSNDBUF** and **DB2_HADR_SORCVBUF** settings. Some operating systems automatically round or silently cap the user specified value.

You can use the HADR simulator (a command-line tool that generates a simulated HADR workload) to measure network performance and to experiment with various network tuning options. You can download the simulator at https://www.ibm.com/developerworks/community/wikis/home/wiki/DB2HADR/page/HADR%20simulator.

Network congestion

For each log write on the primary, the same log pages are also sent to the standby. Each write operation is called a *flush*. The size of the flush is limited to the log buffer size on the primary database (which is controlled by the database configuration parameter **logbufsz**). The exact size of each flush is nondeterministic. A larger log buffer does not necessarily lead to a larger flush size.

If the standby database is too slow replaying log pages, its log-receiving buffer might fill up, thereby preventing the buffer from receiving more log pages. In SYNC and NEARSYNC modes, if the primary database flushes its log buffer one more time, the data is likely to be buffered in the network pipeline consisting of the primary machine, the network, and the standby database. Because the standby database does not have free buffer to receive the data, it cannot acknowledge, so the primary database becomes blocked while it is waiting for the standby database's acknowledgement.

In ASYNC mode, the primary database continues to send log pages until the pipeline fills up and it cannot send additional log pages. This condition is called *congestion*. Congestion is reported by the **hadr_connect_status** monitor element. For SYNC and NEARSYNC modes, the pipeline can usually absorb a single flush and congestion does not occur. However, the primary database remains blocked waiting for an acknowledgment from the standby database on the flush operation.

Congestion can also occur if the standby database is replaying log records that take a long time to replay, such as database or table reorganization log records.

In SUPERASYNC mode, since the transaction commit operations on the primary database are not affected by the relative slowness of the HADR network or the standby HADR server, the log gap between the primary database and the standby database might continue to increase. It is important to monitor the log gap as it is an indirect measure of the potential number of transactions that might be lost should a true disaster occur on the primary system. In disaster recovery scenarios, any transactions that are committed during the log gap would not be available to the standby database. Therefore, monitor the log gap by using the **hadr_log_gap** monitor element; if it occurs that the log gap is not acceptable, investigate the network interruptions or the relative speed of the standby HADR server and take corrective measures to reduce the log gap.

Recommendations for minimizing network congestion:

- The standby database should be powerful enough to replay the logged operations of the database as fast as they are generated on the primary. Identical primary and standby hardware is recommended.
- Consider tuning the size of the standby database log-receiving buffer by using the **DB2_HADR_BUF_SIZE** registry variable.

  A larger buffer can help to reduce congestion, although it might not remove all of the causes of congestion. By default, the size of the standby database log-receiving buffer is two times the size of the primary database log-writing buffer. The database configuration parameter **logbufsz** specifies the size of the primary database log-writing buffer.

  You can determine if the standbys log-receiving buffer is inadequate by using the **db2pd** command with the `-hadr` option or the MON_GET_HADR table function. If the value for the `STANDBY_RECV_BUF_PERCENT` field, which indicates the percentage of standby log receiving buffer that is being used, is close to 100, increase the **DB2_HADR_BUF_SIZE** setting.
- Consider setting the **DB2_HADR_PEER_WAIT_LIMIT** registry variable, which allows you to prevent primary database logging from blocking because of a slow or blocked standby database.

  When the **DB2_HADR_PEER_WAIT_LIMIT** registry variable is set, the HADR primary database breaks out of the peer state if logging on the primary database is blocked for the specified number of seconds because of log replication to the standby. When this limit is reached, the primary database breaks the connection to the standby database. If the peer window is disabled, the primary enters disconnected state and logging resumes. If the peer window is enabled, the primary database enters disconnected peer state, in which logging continues to be blocked. The primary database leaves disconnected peer state upon re-connection or peer window expiration. Logging resumes after the primary database leaves disconnected peer state.

Note: If you set **`DB2_HADR_PEER_WAIT_LIMIT`**, use a minimum value of `10` to avoid triggering false alarms.

Honoring peer window transition when breaking out of peer state ensures peer window semantics for safe takeover in all cases. If the primary fails during the transition, normal peer window protection still applies (safe takeover from standby as long as it is still in disconnected-peer state).

- In most systems, the logging capability is not driven to its limit. Even in SYNC mode, there might not be an observable slow down on the primary database. For example, if the limit of logging is 40 MB per second with HADR enabled, but the system was just running at 30 MB per second before HADR is enabled, then you might not notice any difference in overall system performance.
- To speed up the catchup process, you can use a shared log archive device. However, if the shared device is a serial device such as a tape drive, you might experience performance degradation on both the primary and standby databases because of mixed read and write operations.
- If you are going to use the reads on standby feature, the standby must have the resources to accommodate this additional work.
- If you are going to use the reads on standby feature, configure your buffer pools on the primary, and that information is shipped to the standby through logs.
- If you are going to use the reads on standby feature, Tune the **`pckcachesz`**, **`catalogcache_sz`**, **`applheapsz`**, and **`sortheap`** configuration parameters on the standby.

Cluster managers and high availability disaster recovery (HADR)

You can implement DB2 High Availability Disaster Recovery (HADR) databases on nodes of a cluster, and use a cluster manager to improve the availability of your database solution.

You can have both the primary database and the standby database managed by the same cluster manager, or you can have the primary database and the standby database managed by different cluster managers.

Set up an HADR pair where the primary and standby databases are serviced by the same cluster manager

This configuration is best suited to environments where the primary and standby databases are located at the same site and where the fastest possible failover is required. These environments would benefit from using HADR to maintain DBMS availability, rather using crash recovery or another recovery method.

You can use the cluster manager to quickly detect a problem and to initiate a takeover operation. Because HADR requires separate storage for the DBMS, the cluster manager should be configured with separate volume control. This configuration prevents the cluster manager from waiting for failover to occur on the volume before using the DBMS on the standby system. You can use the automatic client reroute feature to redirect client applications to the new primary database.

Set up an HADR pair where the primary and standby databases are not serviced by the same cluster manager

This configuration is best suited to environments where the primary and standby databases are located at different sites and where high availability is required for disaster recovery in the event of a complete site failure. There are several ways you can implement this configuration. When an HADR primary or standby database is part of a cluster, there are two possible failover scenarios.

- If a partial site failure occurs and a node to which the DBMS can fail over remains available, you can choose to perform a cluster failover. In this case, the IP address and volume failover is performed using the cluster manager; HADR is not affected.
- If a complete site failure occurs where the primary database is located, you can use HADR to maintain DBMS availability by initiating a takeover operation. If a complete site failure occurs where the standby database is located, you can repair the site or move the standby database to another site.

Performing an HADR failover operation

When you want the current standby database to become the new primary database because the current primary database is not available, you can perform a failover.

About this task

Warning:

This procedure might cause a loss of data. Review the following information before performing this emergency procedure:

- Ensure that the primary database is no longer processing database transactions. If the primary database is still running, but cannot communicate with the standby database, executing a forced takeover operation (issuing the **TAKEOVER HADR** command with the **BY FORCE** option) could result in two primary databases. When there are two primary databases, each database will have different data, and the two databases can no longer be automatically synchronized.
  - Deactivate the primary database or stop its instance, if possible. (This might not be possible if the primary system has hung, crashed, or is otherwise inaccessible.) After a takeover operation is performed, if the failed database is later restarted, it will not automatically assume the role of primary database.
- The likelihood and extent of transaction loss depends on your specific configuration and circumstances:
  - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is synchronous (SYNC), the standby database will not lose transactions that were reported committed to an application before the primary database failed.
  - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is near synchronous (NEARSYNC), the standby database can only lose transactions committed by the primary database if both the primary and the standby databases fail at the same time.
  - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is asynchronous (ASYNC), the standby database can lose transactions committed by the primary database if the standby database did not receive all of the log records for the transactions before the takeover operation was performed. The standby database can also

lose transactions committed by the primary database if the standby database crashes before it was able to write all the received logs to disk.

  Note: Peer window is not allowed in ASYNC mode, therefore the primary database will never enter disconnected peer state in that mode.

  - If the primary database fails while in remote catchup state and the synchronization mode is super asynchronous (SUPERASYNC), the standby database can lose transactions committed by the primary database if the standby database did not receive all of the log records for the transactions before the takeover operation was performed. The standby database can also lose transactions committed by the primary database if the standby database crashes before it was able to write all the received logs to disk.

    Note: Databases can never be in peer or disconnected peer state in SUPERASYNC mode. Failover (forced takeover) is allowed in remote catchup state only if the synchronization mode is SUPERASYNC.
  - If the primary database fails while in remote catchup pending state, transactions that have not been received and processed by the standby database will be lost.

    Note: Any log gap shown in the database snapshot will represent the gap at the last time the primary and standby databases were communicating with each other; the primary database might have processed a very large number of transactions since that time.
- Ensure that any application that connects to the new primary (or that is rerouted to the new primary by client reroute), is prepared to handle the following:
  - There is data loss during failover. The new primary does not have all of the transactions committed on the old primary. This can happen even when the **hadr_syncmode** configuration parameter is set to SYNC. Because an HADR standby applies logs sequentially, you can assume that if a transaction in an SQL session is committed on the new primary, all previous transactions in the same session have also been committed on the new primary. The commit sequence of transactions across multiple sessions can be determined only with detailed analysis of the log stream.
  - It is possible that a transaction can be issued to the original primary, committed on the original primary and replicated to the new primary (original standby), but not be reported as committed because the original primary crashed before it could report to the client that the transaction was committed. Any application you write should be able to handle that transactions issued to the original primary, but not reported as committed on the original primary, are committed on the new primary (original standby).
  - Some operations are not replicated, such as changes to database configuration and to external UDF objects.
- The **TAKEOVER HADR** command can only be issued on the standby database.
- HADR does not interface with the DB2 fault monitor (db2fm) which can be used to automatically restart a failed database. If the fault monitor is enabled, you should be aware of possible fault monitor action on a presumably failed primary database.
- A takeover operation can only take place if the primary and standby databases are in peer state or the standby database is in remote catchup pending state. If the standby database is in any other state, an error will be returned.

  Note: You can make a standby database that is in local catchup state available for normal use by converting it to a standard database. To do this, shut the database down by issuing the **DEACTIVATE DATABASE** command, and then issue the **STOP HADR** command. Once HADR has been stopped, you must complete a rollforward operation on the former standby database before it can be used. A database cannot rejoin an HADR pair after it has been converted from a standby database to a standard database. To restart HADR on the two servers, follow the procedure for initializing HADR.

If you have configured a peer window, shut down the primary before the window expires to avoid potential transaction loss in any related failover.

In a failover scenario, a takeover operation can be performed through the command line processor (CLP), or the db2HADRTakeover application programming interface (API).

Procedure

The following procedure shows you how to initiate a failover on the primary or standby database using the CLP:

1. Completely disable the failed primary database. When a database encounters internal errors, normal shutdown commands might not completely shut it down. You might need to use operating system commands to remove resources such as processes, shared memory, or network connections.
2. Issue the **TAKEOVER HADR** command with the **BY FORCE** option on the standby database. In the following example the failover takes place on database LEAFS:

   ```
   TAKEOVER HADR ON DB LEAFS BY FORCE
   ```

   The **BY FORCE** option is required because the primary is expected to be offline.

   If the primary database is not completely disabled, the standby database will still have a connection to the primary and will send a message to the primary database asking it to shutdown. The standby database will still switch to the role of primary database whether or not it receives confirmation from that the primary database has been shutdown.

Switching database roles in high availability disaster recovery (HADR)

During high availability disaster recovery (HADR), use the **TAKEOVER HADR** command to switch the roles of the primary and standby databases.

About this task

- The **TAKEOVER HADR** command can only be issued on the standby database. If the primary database is not connected to the standby database when the command is issued, the takeover operation will fail.
- The **TAKEOVER HADR** command can only be used to switch the roles of the primary and standby databases if the databases are in peer state. If the standby database is in any other state, an error message will be returned.

Procedure

To switch the HADR database roles:

- Use the CLP to initiate a takeover operation on the standby database, issue the **TAKEOVER HADR** command without the **BY FORCE** option on the standby database.

  In the following example, the takeover operation takes place on the standby database LEAFS:

  ```
  TAKEOVER HADR ON DB LEAFS
  ```

A log full error is slightly more likely to occur immediately following a takeover operation. To limit the possibility of such an error, an asynchronous buffer pool flush is automatically started at the end of each takeover. The likelihood of a log full error decreases as the asynchronous buffer pool flush progresses. Additionally, if your configuration provides a sufficient amount of active log space, a log full error is even more unlikely. If a log full error does occur, the current transaction is aborted and rolled back.

Note: Issuing the **TAKEOVER HADR** command without the **BY FORCE** option will cause any applications currently connected to the HADR primary database to be forced off. This action is designed to work in coordination with automatic client reroute to assist in rerouting clients to the new HADR primary database after a role switch. However, if the forcing off of applications from the primary would be disruptive in your environment, you might want to implement your own procedure to shut down such applications prior to performing a role switch, and then restart them with the new HADR primary database as their target after the role switch is completed.

- Call the db2HADRTakeover application programming interface (API) from an application.
- Open the task assistant for the **TAKEOVER HADR** command in IBM Data Studio.

Reintegrating a database after a takeover operation

If you executed a takeover operation in a high availability disaster recovery (HADR) environment because the primary database failed, you can bring the failed database back online and use it as a standby database or return it to its status as primary database.

Procedure

To reintegrate the failed primary database into the HADR pair as the new standby database:

1. Repair the system where the original primary database resided. This could involve repairing failed hardware or rebooting the crashed operating system.
2. Restart the failed primary database as a standby database. In the following example, database LEAFS is started as a standby database:

   ```
   START HADR ON DB LEAFS AS STANDBY
   ```

   Note: Reintegration will fail if the two copies of the database have incompatible log streams. In particular, HADR requires that the original primary database did not apply any logged operation that was never reflected on the original standby database before it took over as the new primary database. If this did occur, you can restart the original primary database as a standby database by restoring a backup image of the new primary database or by initializing a split mirror.

   Successful return of this command does not indicate that reintegration has succeeded; it means only that the database has been started. Reintegration is still in progress. If reintegration subsequently fails, the database will shut itself down. You should monitor standby states using the **GET SNAPSHOT FOR DATABASE** command or the **db2pd** tool to make sure that the standby database stays online and proceeds with the normal state transition. If necessary, you can check the administration notification log file and the **db2diag** log file to find out the status of the database.

What to do next

After the original primary database has rejoined the HADR pair as the standby database, you can choose to perform a failback operation to switch the roles of the databases to enable the original primary database to be once again the primary database. To perform this failback operation, issue the following command on the standby database:

```
TAKEOVER HADR ON DB LEAFS
```

Note:

1. If the HADR databases are not in peer state or the pair is not connected, this command will fail.
2. Open sessions on the primary database are forced closed and inflight transactions are rolled back.
3. When switching the roles of the primary and standby databases, the **BY FORCE** option of the **TAKEOVER HADR** command cannot be specified.

Monitoring high availability disaster recovery (HADR) environments

Monitoring is an integral part of setting up and maintaining your HADR setup. The DB2 monitoring interfaces provide a detailed picture of the configuration and health of your environment.

You can use a number of methods to monitor the status of your HADR databases. There are two preferred ways of monitoring HADR:

- The db2pd command
- The MON_GET_HADR table function

You can also use the following methods, but starting in Version 10.1, they are deprecated, and they might be removed in a future release:

- The **GET SNAPSHOT FOR DATABASE** command
- The db2GetSnapshot API
- The SNAPHADR administrative view
- The SNAP_GET_HADR table function
- Other snapshot administrative views and table functions

db2pd command

This command retrieves information from the DB2 memory sets. You can issue this command from either a primary database or a standby database. If you are using multiple standby mode and you issue this command from a standby, it does not return any information about the other standbys. If you issue this command from the primary, it returns information on all standbys

To view information about high availability disaster recovery for database HADRDB, you could issue the following command:

```
db2pd -db HADRDB -hadr
```

Assuming you issued that command from the primary, you would receive something like the following sample output:

```
  Database Member 0 -- Database HADRDB -- Active -- Up 0 days 00:23:17 --
Date 06/08/2011 13:57:23

                            HADR_ROLE = PRIMARY
                          REPLAY_TYPE = PHYSICAL
```

```
                        HADR_SYNCMODE = SYNC
                           STANDBY_ID = 1
                        LOG_STREAM_ID = 0
                           HADR_STATE = PEER
                  PRIMARY_MEMBER_HOST = hostP.ibm.com
                     PRIMARY_INSTANCE = db2inst
                       PRIMARY_MEMBER = 0
                  STANDBY_MEMBER_HOST = hostS1.ibm.com
                     STANDBY_INSTANCE = db2inst
                       STANDBY_MEMBER = 0
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
          HEARTBEAT_INTERVAL(seconds) = 25
                HADR_TIMEOUT(seconds) = 100
        TIME_SINCE_LAST_RECV(seconds) = 3
             PEER_WAIT_LIMIT(seconds) = 0
           LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
   LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
                  LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
                  HADR_LOG_GAP(bytes) = 0
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
       STANDBY_RECV_REPLAY_GAP(bytes) = 0
                     PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                     STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
              STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
         STANDBY_RECV_BUF_SIZE(pages) = 16
             STANDBY_RECV_BUF_PERCENT = 0
           STANDBY_SPOOL_LIMIT(pages) = 0
                 PEER_WINDOW(seconds) = 0
             READS_ON_STANDBY_ENABLED = Y
    STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N
```

MON_GET_HADR table function

If you issue this query on the primary, it will return information on all standbys. If you want to issue the MON_GET_HADR function against a standby database, be aware of the following points:

- You must enable reads on standby on the standby.
- Even if your HADR setup is in multiple standby mode, the table function does not return any information about any other standbys.

For example, you could issue the following query on the primary database:

```
db2 "select HADR_ROLE, STANDBY_ID, HADR_STATE,
            varchar(PRIMARY_MEMBER_HOST,20) as PRIMARY_MEMBER_HOST,
            varchar(STANDBY_MEMBER_HOST,20) as STANDBY_MEMBER_HOST
     from table (mon_get_hadr(NULL))"
```

Sample output is as follows:

```
HADR_ROLE  STANDBY_ID  HADR_STATE  PRIMARY_MEMBER_HOST  STANDBY_MEMBER_HOST
---------  ----------  ----------  -------------------  -------------------
PRIMARY             1  PEER        hostP.ibm.com        hostS1.ibm.com

1 record(s) selected.
```

GET SNAPSHOT FOR DATABASE command

This command collects status information and formats the output. The information that is returned is a snapshot of the database manager operational status at the time that you issued the command. HADR information is displayed in the output under the heading `HADR status`.

db2GetSnapshot API

This API collects database manager monitor information and writes it to a user-allocated data buffer. The information that is returned is a snapshot of the database manager operational status at the time that the API was called.

SNAPHADR administrative view and SNAP_GET_HADR table function

This administrative view and this table function return information about HADR from a database snapshot, in particular, the HADR logical data group.

Other snapshot administrative views and table functions

You can use the following snapshot administrative views and table functions, which are not HADR specific and return a wider set of information, to query a subsection of the HADR information:

- ADMIN_GET_STORAGE_PATHS
- MON_GET_TRANSACTION_LOG
- SNAPDB
- SNAPDB_MEMORY_POOL
- SNAPDETAILLOG
- SNAP_GET_DB
- SNAP_GET_DB_MEMORY_POOL

Stopping DB2 High Availability Disaster Recovery (HADR)

If you are using the DB2 High Availability Disaster Recovery (HADR) feature, stopping HADR operations to perform maintenance on the primary or standby databases might be necessary. Stop HADR operations only on the database that you are performing maintenance. To stop using HADR completely, stop HADR on both databases.

About this task

Warning: If you want to stop the specified database but you still want it to maintain its role as either an HADR primary or standby database, do not issue the STOP HADR command. If you issue the **STOP HADR** command the database will become a standard database and might require reinitialization in order to resume operations as an HADR database. Instead, issue the **DEACTIVATE DATABASE** command.

If you issue the **STOP HADR** command against a standard database, an error will be returned.

Procedure

To stop HADR operations on the primary or standby database:

- From the CLP, issue the **STOP HADR** command on the database where you want to stop HADR operations.

  In the following example, HADR operations are stopped on database SOCKS:

  ```
  STOP HADR ON DATABASE SOCKS
  ```

  If you issue this command against an inactive primary database, the database switches to a standard database and remains offline.

  If you issue this command against an inactive standby database the database switches to a standard database, is placed in rollforward pending state, and remains offline.

If you issue this command on an active primary database, logs stop being shipped to the standby database and all HADR engine dispatchable units (EDUs) are shut down on the primary database. The database switches to a standard database and remains online. Transaction processing can continue. You can issue the START HADR command with the AS PRIMARY option to switch the role of the database back to primary database.

If you issue this command on an active standby database, an error message is returned, indicating that you must deactivate the standby database before attempting to convert it to a standard database.

- From an application, call the **db2HADRStop** application programming interface (API).
- From IBM Data Studio, open the task assistant for the **STOP HADR** command.

Chapter 50. DB2 high availability instance configuration utility (db2haicu)

DB2 high availability instance configuration utility (db2haicu) is a text-based utility that you can use to configure and administer your highly available databases in a clustered environment.

db2haicu collects information about your database instance, your cluster environment, and your cluster manager by querying your system. You supply more information through parameters to the **db2haicu** call, an input file, or at run time by providing information at **db2haicu** prompts.

Syntax

```
db2haicu [ -f XML-input-file-name ]
         [ -disable ]
         [ -delete [ dbpartitionnum db-partition-list |
                     hadrdb database-name ] ]
```

Parameters

The parameters that you pass to the **db2haicu** command are case-sensitive, and must be in lower case.

-f *XML-input-file-name*

You can use the **-f** parameter to specify your cluster domain details in an XML input file, *XML-input-file-name*. For more information, see: "Running db2haicu with an XML input file" on page 934.

-disable

A database manager instance is considered *configured for high availability* after you use **db2haicu** to create a cluster domain for that instance. When a database manager instance is configured for high availability, then whenever you perform certain database manager administrative operations that require related cluster configuration changes, the database manager communicates those cluster configuration changes to the cluster manager. When the database manager coordinates these cluster management tasks with the cluster manager for you, you do not have to perform a separate cluster manager operation for those administrative tasks. This integration between the database manager and the cluster manager is a function of the DB2 High Availability Feature.

You can use the **-disable** parameter to cause a database manager instance to cease to be configured for high availability. If the database manager instance is no longer configured for high availability, then the database manager does not coordinate with the cluster manager if you perform any database manager administrative operations that require related cluster configuration changes.

To reconfigure a database manager instance for high availability, you can run **db2haicu** again.

-delete

You can use the **-delete** parameter to delete resource groups for the current database manager instance.

If you do not use either the **dbpartitionnum** parameter or the **hadrdb** parameter, then **db2haicu** removes all the resource groups that are associated with the current database manager instance.

dbpartitionnum *db-partition-list*
: You can use the **dbpartitionnum** parameter to delete resource groups that are associated with the database partitions listed in *db-partition-list*. *db-partition-list* is a comma-separated list of numbers that identify the database partitions.

hadrdb *database-name*
: You can use the **hadrdb** parameter to delete resource groups that are associated with the high availability disaster recovery (HADR) database named *database-name*.

If there are no resource groups that are left in the cluster domain after **db2haicu** removes the resource groups, then **db2haicu** will also remove the cluster domain.

Running **db2haicu** with the **-delete** parameter causes the current database manager instance to cease to be configured for high availability. If the database manager instance is no longer configured for high availability, then the database manager does not coordinate with the cluster manager if you perform any database manager administrative operations that require related cluster configuration changes.

To reconfigure a database manager instance for high availability, you can run **db2haicu** again.

Startup mode

The first time that you run DB2 high availability instance configuration utility (db2haicu) for a given database manager instance, **db2haicu** operates in startup mode.

When you run **db2haicu**, **db2haicu** examines your database manager instance and your system configuration, and searches for an existing *cluster domain*. A cluster domain is a model that contains information about your cluster elements such databases, mount points, and failover policies. You create a cluster domain using DB2 high availability instance configuration utility (db2haicu).

When you run **db2haicu** for a given database manager instance, and there is no cluster domain that is already created and configured for that instance, **db2haicu** will immediately begin the process of creating and configuring a new cluster domain. **db2haicu** creates a new cluster domain by prompting you for information such as a name for the new cluster domain and the hostname of the current machine.

If you create a cluster domain, but do not complete the task of configuring the cluster domain, then the next time you run **db2haicu**, **db2haicu** will resume the task of configuring the cluster domain.

After you create and configure a cluster domain for a database manager instance, **db2haicu** will run in maintenance mode.

Maintenance mode

When you run DB2 high availability instance configuration utility (db2haicu) and there is already a cluster domain created for the current database manager instance, **db2haicu** operates in maintenance mode.

When **db2haicu** is running in maintenance mode, **db2haicu** presents you with a list of configuration and administration tasks that you can perform.

db2haicu maintenance tasks include adding cluster elements such as databases or cluster nodes to the cluster domain, and removing elements from the cluster domain. **db2haicu** maintenance tasks also include modifying the details of cluster domain elements such as the failover policy for the database manager instance.

When you run **db2haicu** in maintenance mode, **db2haicu** presents you with a list of operations you can perform on the cluster domain:

- Add or remove cluster nodes (machine identified by hostname)
- Add or remove a network interface (network interface card)
- Add or remove database partitions (partitioned database environment only)
- Add or remove a DB2 High Availability Disaster Recovery (HADR) database
- Add or remove a highly available database
- Add or remove a mount point
- Add or remove an IP address
- Add or remove a non-critical path
- Move database partitions and HADR databases for scheduled maintenance
- Change failover policy for the current instance
- Create a new quorum device for the cluster domain
- Destroy the cluster domain

Prerequisites

There is a set of tasks you must perform before using DB2 high availability instance configuration utility (db2haicu).

General

Before a database manager instance owner can run **db2haicu**, a user with root authority must run the **preprpnode** command.

preprpnode is part of the Reliable Scalable Cluster Technology (RSCT) fileset for AIX and the RSCT package for Linux. **preprpnode** handles initializing the nodes for intracluster communication. The **preprpnode** command is run as a part of setting up the cluster. For more information about preprpnode, see:

- `preprpnode` Command (AIX)
- `preprpnode` command (Linux)

For more information about RSCT, see RSCT Administration Guide - What is RSCT?

Also, a user with root authority must disable the iTCO_wdt and iTCO_vendor_support modules.

- On SUSE, add the following lines to the `/etc/modprobe.d/blacklist` file:

```
alias iTCO_wdt off
alias iTCO_vendor_support off
```

- On RHEL, add the following lines to the /etc/modprobe.conf file:

  ```
  blacklist iTCO_wdt
  blacklist iTCO_vendor_support
  ```

You can verify that the modules are disabled by using the **lsmod** command.

Before running **db2haicu**, a database manager instance owner must perform the following tasks:

- Synchronize services files on all machines that will be added to the cluster.
- Run the **db2profile** script for the database manager instance that will be used to create the cluster domain.
- Start the database manager using the **db2start** command.

DB2 High Availability Disaster Recovery (HADR)

If you will be using HADR functionality, perform the following tasks:

- Ensure all DB2 High Availability Disaster Recovery (HADR) databases are started in their respective primary and standby database roles, and that all HADR primary-standby database pairs are in peer state.
- Configure **hadr_peer_window** for all HADR databases to a value of at least 120 seconds.
- Disable DB2 fault monitor.

Partitioned database environment

If you have multiple database partitions to configure for high availability, perform the following steps:

- Configure the **DB2_NUM_FAILOVER_NODES** registry variable on all machines that will be added to the cluster domain.
- (Optional) Activate the database before running **db2haicu**.

Configuring a clustered environment

You can configure and administer your databases in a clustered environment using DB2 high availability instance configuration utility (db2haicu). When you specify database manager instance configuration details to **db2haicu**, **db2haicu** communicates the required cluster configuration details to your cluster managing software.

Before you begin

- There is a set of tasks you must perform before using DB2 high availability instance configuration utility (db2haicu). For more information, see: “Prerequisites” on page 929.

About this task

You can run **db2haicu** interactively, or using an XML input file:

Interactive mode

When you invoke DB2 high availability instance configuration utility (db2haicu) by running the **db2haicu** command without specifying an XML input file with the **-f** parameter, the utility runs in interactive mode. In interactive mode, **db2haicu** displays information and queries you for information in a text-based format. For more information, see: "Running db2haicu interactively" on page 933

Batch mode with an XML input file
You can use the `-f` *`input-file-name`* parameter with the **db2haicu** command to run DB2 high availability instance configuration utility (db2haicu) with an XML input file specifying your configuration details. Running **db2haicu** with an XML input file is useful when you must perform configuration tasks multiple times, such as when you have multiple database partitions to be configured for high availability. For more information, see: "Running db2haicu with an XML input file" on page 934

For a detailed scenario that uses **db2haicu** with both methods to set up an HADR pair, see "Automated Cluster Controlled HADR (High Availability Disaster Recovery) Configuration Setup using the IBM DB2 High Availability Instance Configuration Utility (db2haicu)".

DB2 instances that are configured in Version 9.1 for high availability by using the **regdb2salin** script are not supported on Versions 9.5 and later. Use the **db2haicu** utility to configure these instances for high availability.

Restrictions

There are some restrictions for using the DB2 high availability instance configuration utility (db2haicu).

Procedure

Perform the following steps for each database manager instance:

1. Create a new cluster domain.

   When you run DB2 high availability instance configuration utility (db2haicu) for the first time for a database manager instance, **db2haicu** creates a model of your cluster, called a *cluster domain*.
2. Continue to refine the cluster domain configuration, and administer and maintain the cluster domain

   When you are modifying the cluster domain model of your clustered environment using **db2haicu**, the database manager propagates the related changes to your database manager instance and cluster configuration.

Restrictions for db2haicu

There are some restrictions for using the DB2 high availability instance configuration utility (db2haicu).

- "Software and hardware"
- "Configuration tasks" on page 932
- "Usage notes" on page 932
- "Recommendations" on page 933

Software and hardware

- **db2haicu** does not support the configuration of mount resources which are based off of Logical Volume Manager (LVM) on any platform other than AIX.

Configuration tasks

You cannot perform the following tasks using **db2haicu**:

- You cannot configure automatic client reroute using **db2haicu**.
- When you upgrade from DB2 for Linux, UNIX, and Windows Version 10.1 to a later version, you cannot use **db2haicu** to migrate your cluster configuration. To migrate a cluster configuration, you must perform the following steps:
  1. Delete the existing cluster domain (if one exists)
  2. Upgrade the database server
  3. Create a new cluster domain using **db2haicu**

Usage notes

db2haicu is not supported in a DB2 pureScale environment. Use the **db2cluster** command instead to configure clustered environments.

Consider the following **db2haicu** usage notes when planning your cluster configuration and administration activities:

- Even though **db2haicu** performs some administration tasks that normally require root authority, **db2haicu** runs with the privileges of the database manager instance owner. **db2haicu** initialization, which is performed by a root user, enables **db2haicu** to carry out the required configuration changes despite having only instance owner privileges.
- When you create a new cluster domain, **db2haicu** does not verify that the name you specify for the new cluster domain is valid. For example, **db2haicu** does not confirm that the name is a valid length, or contains valid characters, or that is not the same name as an existing cluster domain.
- **db2haicu** does not verify or validate information that a user specifies and that is passed to a cluster manager. Because **db2haicu** cannot be aware of all cluster manager restrictions with respect to cluster object names, for example, **db2haicu** passes text to the cluster manager without validating it for things like valid characters, or length.
- If an error happens and **db2haicu** fails while you are creating and configuring a new cluster domain, you must perform the following steps:
  1. Remove the resource groups of the partially created cluster domain by running **db2haicu** using the **-delete** parameter
  2. Re-create the new cluster domain by calling **db2haicu** again.
- When you run **db2haicu** with the **-delete** parameter, **db2haicu** deletes the resource groups that are associated with the current database manager instance immediately, without confirming whether those resource groups are locked.
- To remove resource groups that are associated with the database manager instances of a DB2 high availability disaster recovery (HADR) database pair, perform the following steps:
  1. Run **db2haicu** with the **-delete** parameter against the database manager instance of the HADR standby database first.
  2. Run **db2haicu** with the **-delete** parameter against the database manager instance of the HADR primary database.
- To remove a virtual IP from an HADR resource group using **db2haicu**, you must remove it from the instance on which it was created.
- The ASYNC and SUPERASYNC HADR synchronization modes are not supported by **db2haicu**.

- If a cluster operation you attempt to perform using **db2haicu** times out, **db2haicu** does not return an error to you. When a cluster operation times out, you do not know that the operation timed out unless you review diagnostic logs after you make the **db2haicu** call; or unless a subsequent cluster action fails, and while you are investigating that subsequent failure, you determine that the original cluster operation timed out.
- If you attempt to change the failover policy for a given database instance to active-passive, there is one condition under which that configuration operation fails, but for which **db2haicu** does not return an error to you. If you specify a machine that is currently offline to be the *active* machine, **db2haicu** does not make that machine the active machine, but **db2haicu** does not return an error that indicates that the change did not succeed.
- For a shared disk configuration, **db2haicu** does not support a nested mount configuration because DB2 does not enforce the disk mount order.
- When you are adding network interface cards (NICs) to a network, you cannot add NICs with different subnet masks to the same network using **db2haicu**. If you want to add NICs with different subnet masks to the same network, use the following SA MP command:

  ```
  mkequ <name> IBM.NetworkInterface:<eth0>:<node0>,...,<ethN>:<nodeN>
  ```
- The enhanced capable option is not available for the Enhanced Concurrent Logical Volume Manager.

Recommendations

The following is a list of recommendations for configuration your cluster, and your database manager instances when you are using **db2haicu**.

- When you add new mount points for the cluster by adding entries to /etc/fstab, use the **noauto** option to prevent the mount points from being automatically mounted on more than one machine in the cluster. For example:

  ```
  dev/vpatha1      /db/svtpdb/NODE0010        ext3  noauto 0 0
  ```

Running db2haicu interactively

When you invoke DB2 high availability instance configuration utility (db2haicu) by running the **db2haicu** command without specifying an XML input file with the **-f** parameter, the utility runs in interactive mode. In interactive mode, **db2haicu** displays information and queries you for information in a text-based format.

Before you begin

- There is a set of tasks you must perform before using DB2 high availability instance configuration utility (db2haicu). For more information, see: "Prerequisites" on page 929.

About this task

When you run **db2haicu** in interactive mode, you see information and questions presented to you in text format on your screen. You can enter the information requested by **db2haicu** at a prompt at the bottom of your screen.

Procedure

To run **db2haicu** in interactive mode, call the **db2haicu** command without the -f *input-file-name*.

Running db2haicu with an XML input file

Use the db2haicu command with the -f input-file-name parameter to specify an XML input file with your configuration details. Using an XML input file is useful when you must perform configuration tasks multiple times, such as configuring multiple database partitions for high availability.

Before you begin

- There is a set of tasks you must perform before using DB2 high availability instance configuration utility (db2haicu). For more information, see: "Prerequisites" on page 929.

About this task

There is a set of sample XML input files located in the `samples` subdirectory of the `sqllib` directory that you can modify and use with **db2haicu** to configure your clustered environment. For more information, see: "Sample XML input files" on page 937

For a detailed scenario that uses **db2haicu** with a sample XML input file to set up an HADR pair, see "Automated Cluster Controlled HADR (High Availability Disaster Recovery) Configuration Setup using the IBM DB2 High Availability Instance Configuration Utility (db2haicu)".

Procedure

1. Create an XML input file. You will use the same XML file if you are configuring database partitions or, in an HADR setup, both the primary and the standby.
2. Call **db2haicu** with the `-f` *input-file-name*. In an HADR setup,
   a. Log on to the standby instance and issue the command.
   b. After **db2haicu** exits, log on to the primary instance and issue the command.

Input file XML schema (DB2ClusterType)

The DB2 high availability instance configuration utility (db2haicu) input file XML schema definition (XSD) defines the cluster domain objects that you can specify in a **db2haicu** XML input file. This **db2haicu** XSD is located in the file called `db2ha.xsd` in the `sqllib/samples/ha/xml` directory.

DB2ClusterType

The root element of the **db2haicu** XML schema definition (XSD) is `DB2Cluster`, which is of type `DB2ClusterType`. A **db2haicu** XML input file must begin with a `DB2Cluster` element.

"XML schema definition"
"Subelements" on page 935
"Attributes" on page 936
"Usage notes" on page 936

XML schema definition

```
<xs:complexType name='DB2ClusterType'>
  <xs:sequence>
    <xs:element name='DB2ClusterTemplate'
                type='DB2ClusterTemplateType'
                minOccurs='0'
                maxOccurs='unbounded'/>
```

```
      <xs:element name='ClusterDomain'
                  type='ClusterDomainType'
                  maxOccurs='unbounded'/>
      <xs:element name='FailoverPolicy'
                  type='FailoverPolicyType'
                  minOccurs='0'/>
      <xs:element name='DB2PartitionSet'
                  type='DB2PartitionSetType'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
      <xs:element name='HADRDBSet'
                  type='HADRDBType'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
      <xs:element name='HADBSet'
                  type='HADBType'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
    </xs:sequence>
    <xs:attribute name='clusterManagerName' type='xs:string' use='optional'/>
</xs:complexType>
```

Subelements

DB2ClusterTemplate

Type: DB2ClusterTemplateType

Usage notes:
Do not include a DB2ClusterTemplateType element in your **db2haicu** XML input file. The DB2ClusterTemplateType element is currently reserved for future use.

ClusterDomain

Type: ClusterDomainType

A ClusterDomainType element contains specifications about: the machines or computers in the cluster domain (also called *cluster domain nodes*); the *network equivalencies* (groups of networks that can fail over for one another); and the *quorum device* (tie-breaking mechanism).

Occurrence rules:
You must include one or more ClusterDomain element in your DB2ClusterType element.

FailoverPolicy

Type: FailoverPolicyType

A FailoverPolicyType element specifies the *failover policy* that the cluster manager should use with the cluster domain.

Occurrence rules:
You can include zero or one FailoverPolicy element in your DB2ClusterType element.

DB2PartitionSet

Type: DB2PartitionSetType

A DB2PartitionSetType element contains information about database partitions. The DB2PartitionSetType element is only applicable in a partitioned database environment.

Occurrence rules:
You can include zero or more DB2PartitionSet elements in your DB2ClusterType element, according to the **db2haicu** db2haicu XML schema definition.

HADRDBSet

Type: HADRDBType

A HADRDBType element contains a list of High Availability Disaster Recovery (HADR) primary and standby database pairs.

Occurrence rules:
You can include zero or more HADRDBSet elements in your DB2ClusterType element, according to the **db2haicu** db2haicu XML schema definition.

Usage notes:

- You must not include HADRDBSet in a partitioned database environment.
- If you include HADRDBSet, then you must specify a failover policy of HADRFailover in the FailoverPolicy element.

HADBSet

Type: HADBType

A HADBType element contains a list of databases to include in the cluster domain, and to make highly available.

Occurrence rules:
You can include zero or more HADBSet elements in your DB2ClusterType element, according to the **db2haicu** db2haicu XML schema definition.

Attributes

clusterManagerName (optional)
The clusterManagerName attribute specifies the cluster manager.

Valid values for this attribute are specified in the following table:

Table 134. Valid values for the clusterManager attribute

clusterManagerName value	**Cluster manager product**
TSA	IBM Tivoli System Automation for Multiplatforms (SA MP)

Usage notes

In a single partition database environment, you will usually only create a single cluster domain for each database manager instance.

One possible configuration for a multi-partition database environment is:

- Set the FailoverPolicy element to Mutual
- In the DB2Partition subelement of DB2PartitionSet, use the MutualPair element to specify two cluster domain nodes that are in a single cluster domain

Sample XML input files

There is a set of sample XML input files located in the `samples` subdirectory of the `sqllib` directory that you can modify and use with **db2haicu** to configure your clustered environment.

db2ha_sample_DPF_NPlusM.xml

The sample file `db2ha_sample_DPF_NPlusM.xml` is an example of an XML input file that you pass to DB2 high availability instance configuration utility (db2haicu) to specify a new *cluster domain*. `db2ha_sample_DPF_NPlusM.xml` is located in the `sqllib/samples/ha/xml` directory.

Features

The `db2ha_sample_DPF_NPlusM.xml` sample demonstrates how to use **db2haicu** with an XML input file to define a cluster domain with the following details:

- quorum device: network
- computers in the cluster (cluster domain nodes): four
- failover policy: N Plus M
- database partitions: two
- virtual (service) IP addresses: one
- shared mount points for failover: four

XML source

```
<!-- ================================================================= -->
<!-- = Use the DB2 High Availability Instance Configuration Utility  = -->
<!-- = (db2haicu) XML schema definition, db2ha.xsd, and specify      = -->
<!-- = IBM Tivoli System Automation for Multiplatforms (SA MP)       = -->
<!-- = Base Component as the cluster manager.                        = -->
<!-- ================================================================= -->
<DB2Cluster xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="db2ha.xsd"
            clusterManagerName="TSA"
            version="1.0">

  <!-- ================================================================= -->
  <!-- = Create a cluster domain named db2HAdomain.                    = -->
  <!-- ================================================================= -->
  <ClusterDomain domainName="db2HAdomain">

    <!-- =============================================================== -->
    <!-- = Specify a network quorum device (IP address: 19.126.4.5).   = -->
    <!-- = The IP must be pingable at all times by each of the cluster = -->
    <!-- = domain nodes.                                               = -->
    <!-- =============================================================== -->
    <Quorum quorumDeviceProtocol="network" quorumDeviceName="19.126.4.5"/>

    <!-- =============================================================== -->
    <!-- = Create a network named db2_public_network_0 with an IP      = -->
    <!-- = network protocol.                                           = -->
    <!-- = This network contains four computers: hasys01, hasys02,     = -->
    <!-- = hasys03, and hasys04.                                       = -->
    <!-- = Each computer has a network interface card called eth0.     = -->
    <!-- = The IP address of eth0 on hasys01 is 19.126.124.30          = -->
    <!-- = The IP address of eth0 on hasys02 is 19.126.124.31          = -->
    <!-- = The IP address of eth0 on hasys03 is 19.126.124.32          = -->
    <!-- = The IP address of eth0 on hasys04 is 19.126.124.33          = -->
    <!-- =============================================================== -->
    <PhysicalNetwork physicalNetworkName="db2_public_network_0"
                     physicalNetworkProtocol="ip">
      <Interface interfaceName="eth0" clusterNodeName="hasys01">
        <IPAddress baseAddress="19.126.124.30"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>

      <Interface interfaceName="eth0" clusterNodeName="hasys02">
        <IPAddress baseAddress="19.126.124.31"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>

      <Interface interfaceName="eth0" clusterNodeName="hasys03">
        <IPAddress baseAddress="19.126.124.32"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>

      <Interface interfaceName="eth0" clusterNodeName="hasys04">
        <IPAddress baseAddress="19.126.124.33"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>

    </PhysicalNetwork>

    <!-- ============================================================= -->
    <!-- = Create a network named db2_private_network_0 with an IP     = -->
    <!-- = network protocol.                                           = -->
    <!-- = This network contains four computers: hasys01, hasys02,     = -->
    <!-- = hasys03, and hasys04 (same as db2_public_network_0.)        = -->
    <!-- = In addition to eth0, each computer has a network interface  = -->
    <!-- = card called eth1.                                           = -->
    <!-- = The IP address of eth1 on hasys01 is 192.168.23.101         = -->
    <!-- = The IP address of eth1 on hasys02 is 192.168.23.102         = -->
    <!-- = The IP address of eth1 on hasys03 is 192.168.23.103         = -->
    <!-- = The IP address of eth1 on hasys04 is 192.168.23.104         = -->
    <!-- ============================================================= -->
    <PhysicalNetwork physicalNetworkName="db2_private_network_0"
                     physicalNetworkProtocol="ip">

      <Interface interfaceName="eth1" clusterNodeName="hasys01">
        <IPAddress baseAddress="192.168.23.101"
                   subnetMask="255.255.255.0"
                   networkName="db2_private_network_0"/>
      </Interface>

      <Interface interfaceName="eth1" clusterNodeName="hasys02">
        <IPAddress baseAddress="192.168.23.102"
                   subnetMask="255.255.255.0"
                   networkName="db2_private_network_0"/>
      </Interface>

      <Interface interfaceName="eth1" clusterNodeName="hasys03">
        <IPAddress baseAddress="192.168.23.103"
                   subnetMask="255.255.255.0"
                   networkName="db2_private_network_0"/>
      </Interface>

      <Interface interfaceName="eth1" clusterNodeName="hasys04">
        <IPAddress baseAddress="192.168.23.104"
                   subnetMask="255.255.255.0"
                   networkName="db2_private_network_0"/>
      </Interface>

    </PhysicalNetwork>
```

```
    <!-- ================================================================= -->
    <!-- = List the computers (cluster nodes) in the cluster domain.   = -->
    <!-- ================================================================= -->
    <ClusterNode clusterNodeName="hasys01"/>
    <ClusterNode clusterNodeName="hasys02"/>
    <ClusterNode clusterNodeName="hasys03"/>
    <ClusterNode clusterNodeName="hasys04"/>

  </ClusterDomain>


  <!-- =================================================================== -->
  <!-- = The failover policy specifies the order in which the cluster  = -->
  <!-- = domain nodes should fail over.                                = -->
  <!-- =================================================================== -->
  <FailoverPolicy>
     <NPlusM />
  </FailoverPolicy>


  <!-- =================================================================== -->
  <!-- = Specify all the details of the database partitions            = -->
  <!-- =================================================================== -->
  <DB2PartitionSet>

    <DB2Partition dbpartitionnum="0" instanceName="db2inst1">
       <VirtualIPAddress baseAddress="19.126.124.250"
                         subnetMask="255.255.255.0"
                         networkName="db2_public_network_0"/>
       <Mount filesystemPath="/ha_dpf1/db2inst1/NODE0000"/>
       <Mount filesystemPath="/hafs/NODE0000"/>
       <NPlusMNode standbyNodeName="hasys03" />
    </DB2Partition>

    <DB2Partition dbpartitionnum="1" instanceName="db2inst1">
       <Mount filesystemPath="/ha_dpf1/db2inst1/NODE0001"/>
       <Mount filesystemPath="/hafs/NODE0001"/>
       <NPlusMNode standbyNodeName="hasys04" />
    </DB2Partition>

  </DB2PartitionSet>

</DB2Cluster>
```

db2ha_sample_HADR.xml

The sample file db2ha_sample_DPF_HADR.xml is an example of an XML input file that you pass to DB2 high availability instance configuration utility (db2haicu) to specify a new *cluster domain*. db2ha_sample_HADR.xml is located in the sqllib/samples/ha/xml directory.

Features

The db2ha_sample_HADR.xml sample demonstrates how to use **db2haicu** with an XML input file to define a cluster domain with the following details:

- quorum device: network
- computers in the cluster (cluster domain nodes): two
- failover policy: HADR
- database partitions: one
- virtual (service) IP addresses: none
- shared mount points for failover: none

XML source

```
<!-- ================================================================= -->
<!-- = DB2 High Availability configuration schema                    = -->
<!-- = Schema describes the elements of DB2 High Availability        = -->
<!-- = IBM Tivoli System Automation for Multiplatforms (SA MP)       = -->
<!-- = that are used in the configuration of a HA cluster            = -->
<!-- ================================================================= -->
<DB2Cluster xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="db2ha.xsd" cluster ManagerName="TSA" version="1.0">

<!-- ================================================================= -->
  <!-- = ClusterDomain element                                         = -->
  <!-- = This element encapsulates the cluster configuration           = -->
  <!-- = specification                                                 = -->
  <!-- = Creating cluster domain of name db2HAdomain                   = -->
  <!-- = Creating an IP quorum device (IP 19.126.4.5)                  = -->
  <!-- = The IP must be pingable at all times by each of the nodes in  = -->
  <!-- = the cluster domain                                            = -->
<!-- ================================================================= -->
 <ClusterDomain domainName="db2HAdomain">
  <Quorum quorumDeviceProtocol="network" quorumDeviceName="19.126.4.5"/>

<!-- ================================================================= -->
  <!-- = Physical network element                                      = -->
  <!-- = The physical network specifies the network type, protocol     = -->
  <!-- = IP address, subnet mask, and NIC name                         = -->
  <!-- = Define two logical groupings of NICs                          = -->
  <!-- = Define two logical groupings of NICs                          = -->
<!-- ================================================================= -->
  <PhysicalNetwork physicalNetworkName="db2_public_network_0"
 physicalNetworkProtocol="ip">
   <Interface interfaceName="eth0" clusterNodeName="hasys01">
    <IPAddress baseAddress="19.126.52.139"
 subnetMask="255.255.255.0" networkName="db2_public_network_0"/>
   </Interface>
   <Interface interfaceName="eth0" clusterNodeName="hasys02">
    <IPAddress baseAddress="19.126.52.140"
 subnetMask="255.255.255.0" networkName="db2_public_network_0"/>
   </Interface>
  </PhysicalNetwork>

  <PhysicalNetwork physicalNetworkName="db2_private_network_0"
 physicalNetworkProtocol="ip">
   <Interface interfaceName="eth1" clusterNodeName="hasys01">
    <IPAddress baseAddress="192.168.23.101"
 subnetMask="255.255.255.0" networkName="db2_private_network_0"/>
   </Interface>
   <Interface interfaceName="eth1" clusterNodeName="hasys02">
     <IPAddress baseAddress="192.168.23.102"
 subnetMask="255.255.255.0" networkName="db2_private_network_0"/>
   </Interface>
  </PhysicalNetwork>

<!-- =============================================================== -->
  <!-- = ClusterNodeName element                                       = -->
  <!-- = The set of nodes in the cluster domain                        = -->
  <!-- = Here the defined set of nodes in the domain is                = -->
  <!-- = hasys01, hasys02                                              = -->
<!-- =============================================================== -->
  <ClusterNode clusterNodeName="hasys01"/>
  <ClusterNode clusterNodeName="hasys02"/>
 </ClusterDomain>

<!-- ================================================================= -->
  <!-- = Failover policy element                                       = -->
  <!-- = The failover policy specifies the failover order of the       = -->
  <!-- = cluster nodes                                                 = -->
  <!-- = In the current sample the failover policy is to restart       = -->
  <!-- = instance in place (LocalRestart)                              = -->
<!-- ================================================================= -->
 <FailoverPolicy>
  <HADRFailover></HADRFailover>
 </FailoverPolicy>

<!-- ================================================================= -->
  <!-- = DB2 Partition element                                         = -->
  <!-- = The DB2 partition type specifies a DB2 Instance Name,         = -->
  <!-- = partition number                                              = -->
<!-- ================================================================= -->
 <DB2PartitionSet>
  <DB2Partition dbpartitionnum="0" instanceName="db2inst1">
  </DB2Partition>
 </DB2PartitionSet>

<!-- ================================================================= -->
  <!-- = HADRDBSet                                                     = -->
  <!-- = Set of HADR Databases for this instance                       = -->
  <!-- = Specify the databaseName, the name of the local instance on   = -->
  <!-- = this machine controlling the HADR database, the name of the   = -->
  <!-- = remote instance in this HADR pair, the name of the local      = -->
  <!-- = hostname and the remote hostname for the remote instance      = -->
<!-- ================================================================= -->
 <HADRDBSet>
  <HADRDB databaseName="HADRDB" localInstance="db2inst1"
   remoteInstance="db2inst1" localHost="hasys01" remoteHost="hasys02"/>
 </HADRDBSet>
</DB2Cluster>
```

Part 7. Security

DB2 database products provide security features that you can use to protect your sensitive data. With the growing number of both internal and external security threats, it is important to separate the tasks of keeping data secure from the management tasks of administering critical systems.

Chapter 51. DB2 security model

Two modes of security control access to the DB2 database system data and functions. Access to the DB2 database system is managed by facilities that reside outside the DB2 database system (authentication), whereas access within the DB2 database system is managed by the database manager (authorization).

Authentication

Authentication is the process by which a system verifies a user's identity. User authentication is completed by a security facility outside the DB2 database system, through an authentication security plug-in module. A default authentication security plug-in module that relies on operating-system-based authentication is included when you install the DB2 database system. For your convenience, the DB2 database manager also ships with authentication plug-in modules for Kerberos and lightweight directory access protocol (LDAP). To provide even greater flexibility in accommodating your specific authentication needs, you can build your own authentication security plug-in module.

The authentication process produces a DB2 authorization ID. Group membership information for the user is also acquired during authentication. Default acquisition of group information relies on an operating-system based group-membership plug-in module that is included when you install the DB2 database system. If you prefer, you can acquire group membership information by using a specific group-membership plug-in module, such as LDAP.

Authorization

After a user is authenticated, the database manager determines if that user is allowed to access DB2 data or resources. Authorization is the process whereby the DB2 database manager obtains information about the authenticated user, indicating which database operations that user can perform, and which data objects that user can access.

The different sources of permissions available to an authorization ID are as follows:

1. Primary permissions: those granted to the authorization ID directly.
2. Secondary permissions: those granted to the groups and roles in which the authorization ID is a member.
3. Public permissions: those granted to PUBLIC.
4. Context-sensitive permissions: those granted to a trusted context role.

Authorization can be given to users in the following categories:

- System-level authorization

  The system administrator (SYSADM), system control (SYSCTRL), system maintenance (SYSMAINT), and system monitor (SYSMON) authorities provide varying degrees of control over instance-level functions. Authorities provide a way both to group privileges and to control maintenance and utility operations for instances, databases, and database objects.
- Database-level authorization

  The security administrator (SECADM), database administrator (DBADM), access control (ACCESSCTRL), data access (DATAACCESS), SQL administrator

(SQLADM), workload management administrator (WLMADM), and explain (EXPLAIN) authorities provide control within the database. Other database authorities include LOAD (ability to load data into a table), and CONNECT (ability to connect to a database).

- Object-level authorization

  Object level authorization involves checking privileges when an operation is performed on an object. For example, to select from a table a user must have SELECT privilege on a table (as a minimum).
- Content-based authorization

  Views provide a way to control which columns or rows of a table specific users can read. Label-based access control (LBAC) determines which users have read and write access to individual rows and individual columns.

You can use these features, in conjunction with the DB2 audit facility for monitoring access, to define and manage the level of security your database installation requires.

Chapter 52. Authentication methods for your server

Access to an instance or a database first requires that the user be *authenticated*. The *authentication type* for each instance determines how and where a user will be verified.

The authentication type is stored in the configuration file at the server. It is initially set when the instance is created. There is one authentication type per instance, which covers access to that database server and all the databases under its control.

If you intend to access data sources from a federated database, you must consider data source authentication processing and definitions for federated authentication types.

Note: You can check the following website for certification information about the cryptographic routines used by the DB2 database management system to perform encryption of the user ID and password when using SERVER_ENCRYPT authentication, and of the user ID, password, and user data when using DATA_ENCRYPT authentication: http://www.ibm.com/security/standards/st_evaluations.shtml.

Switching User on an Explicit Trusted Connection

For CLI/ODBC and XA CLI/ODBC applications, the authentication mechanism used when processing a switch user request that requires authentication is the same as the mechanism used to originally establish the trusted connection itself. Therefore, any other negotiated security attributes (for example, encryption algorithm, encryption keys, and plug-in names) used during the establishment of the explicit trusted connection are assumed to be the same for any authentication required for a switch user request on that trusted connection. Java applications allow the authentication method to be changed on a switch user request (by use of a datasource property).

Because a trusted context object can be defined such that switching user on a trusted connection does *not* require authentication, in order to take full advantage of the switch user on an explicit trusted connection feature, user-written security plug-ins must be able to:

- Accept a user ID-only token
- Return a valid DB2 authorization ID for that user ID

Note: An explicit trusted connection cannot be established if the CLIENT type of authentication is in effect.

Authentication types provided

The following authentication types are provided:

SERVER

Specifies that authentication occurs on the server through the security mechanism in effect for that configuration, for example, through a security plug-in module. The default security mechanism is that if a user ID and password are specified during the connection or attachment attempt, they are sent to the server and compared to the valid user ID and password combinations at the server to determine if the user is permitted to access the instance.

Note: The server code detects whether a connection is local or remote. For local connections, when authentication is SERVER, a user ID and password are not required for authentication to be successful.

SERVER_ENCRYPT

Specifies that the server accepts encrypted SERVER authentication schemes. If the client authentication is not specified, the client is authenticated using the method selected at the server. The user ID and password are encrypted when they are sent over the network from the client to the server.

When the resulting authentication method negotiated between the client and server is SERVER_ENCRYPT, you can choose to encrypt the user ID and password using an AES (Advanced Encryption Standard) 256-bit algorithm. To do this, set the **`alternate_auth_enc`** database manager configuration parameter. This configuration parameter has three settings:

- `NOT_SPECIFIED` (default) means that the server accepts the encryption algorithm that the client proposes, including an AES 256-bit algorithm.
- `AES_CMP` means that if the connecting client proposes DES but supports AES encryption, the server renegotiates for AES encryption.
- `AES_ONLY` means that the server accepts only AES encryption. If the client does not support AES encryption, the connection is rejected.

AES encryption can be used only when the authentication method negotiated between the client and server is SERVER_ENCRYPT.

CLIENT

Specifies that authentication occurs on the database partition where the application is invoked using operating system security. The user ID and password specified during a connection or attachment attempt are compared with the valid user ID and password combinations on the client node to determine whether the user ID is permitted access to the instance. No further authentication will take place on the database server. This is sometimes called single signon.

If the user performs a local or client login, the user is known only to that local client workstation.

If the remote instance has CLIENT authentication, two other parameters determine the final authentication type: **`trust_allclnts`** and **`trust_clntauth`**.

CLIENT level security for TRUSTED clients only:

Trusted clients are clients that have a reliable, local security system.

When the authentication type of CLIENT has been selected, an additional option might be selected to protect against clients whose operating environment has no inherent security.

To protect against unsecured clients, the administrator can select Trusted Client Authentication by setting the **`trust_allclnts`** parameter to `NO`. This implies that all trusted platforms can authenticate the user on behalf of the server. Untrusted clients are authenticated on the Server and must provide a user ID and password. You use the **trust_allclnts** configuration parameter to indicate whether you are trusting clients. The default for this parameter is YES.

Note: It is possible to trust all clients (**trust_allclnts** is YES) yet have some of those clients as those who do not have a native safe security system for authentication.

You might also want to complete authentication at the server even for trusted clients. To indicate where to validate trusted clients, you use the **trust_clntauth** configuration parameter. The default for this parameter is CLIENT.

Note: For trusted clients only, if no user ID or password is explicitly provided when attempting to CONNECT or **ATTACH**, then validation of the user takes place at the client. The **trust_clntauth** parameter is only used to determine where to validate the information provided on the USER or USING clauses.

To protect against all clients, including JCC type 4 clients on z/OS and System i but excluding native DB2 clients on z/OS, OS/390, VM, VSE, and System i, set the **trust_allclnts** parameter to DRDAONLY. Only these clients can be trusted to perform client-side authentication. All other clients must provide a user ID and password to be authenticated by the server.

The **trust_clntauth** parameter is used to determine where the clients mentioned previously are authenticated: if **trust_clntauth** is CLIENT, authentication takes place at the client. If **trust_clntauth** is SERVER, authentication takes place at the client when no user ID and password are provided and at the server when a user ID and password are provided.

Table 135. Authentication Modes using TRUST_ALLCLNTS and TRUST_CLNTAUTH Parameter Combinations.

trust_ allclnts	trust_ clntauth	Untrusted non-DRDA Client Authentication (no user ID & password)	Untrusted non-DRDA Client Authentication (with user ID & password)	Trusted non-DRDA Client Authentication (no user ID & password)	Trusted non-DRDA Client Authentication (with user ID & password)	DRDA Client Authentication (no user ID & password)	DRDA Client Authentication (with user ID & password)
YES	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT
YES	SERVER	CLIENT	SERVER	CLIENT	SERVER	CLIENT	SERVER
NO	CLIENT	SERVER	SERVER	CLIENT	CLIENT	CLIENT	CLIENT
NO	SERVER	SERVER	SERVER	CLIENT	SERVER	CLIENT	SERVER
DRDAONLY	CLIENT	SERVER	SERVER	SERVER	SERVER	CLIENT	CLIENT
DRDAONLY	SERVER	SERVER	SERVER	SERVER	SERVER	CLIENT	SERVER

DATA_ENCRYPT

The server accepts encrypted SERVER authentication schemes and the encryption of user data. The authentication works the same way as that shown with SERVER_ENCRYPT. The user ID and password are encrypted when they are sent over the network from the client to the server.

The following user data are encrypted when using this authentication type:

- SQL and XQuery statements.
- SQL program variable data.
- Output data from the server processing of an SQL or XQuery statement and including a description of the data.
- Some or all of the answer set data resulting from a query.
- Large object (LOB) data streaming.
- SQLDA descriptors.

DATA_ENCRYPT_CMP

The server accepts encrypted SERVER authentication schemes and the encryption of user data. In addition, this authentication type allows compatibility with down level products not supporting DATA_ENCRYPT authentication type. These products are permitted to connect with the SERVER_ENCRYPT authentication type and without encrypting user data. Products supporting the new authentication type must use it. This authentication type is only valid in the server's database manager configuration file and is not valid when used on the **CATALOG DATABASE** command.

KERBEROS

Used when both the DB2 client and server are on operating systems that support the Kerberos security protocol. The Kerberos security protocol performs authentication as a third party authentication service by using conventional cryptography to create a shared secret key. This key becomes a user's credential and is used to verify the identity of users during all occasions when local or network services are requested. The key eliminates the need to pass the user name and password across the network as clear text. Using the Kerberos security protocol enables the use of a single sign-on to a remote DB2 database server. The KERBEROS authentication type is supported on various operating systems.

Kerberos authentication works as follows:

1. A user logging on to the client machine using a domain account authenticates to the Kerberos key distribution center (KDC) at the domain controller. The key distribution center issues a ticket-granting ticket (TGT) to the client.
2. During the first phase of the connection the server sends the target principal name, which is the service account name for the DB2 database server service, to the client. Using the server's target principal name and the target-granting ticket, the client requests a service ticket from the ticket-granting service (TGS) which also resides at the domain controller. If both the client's ticket-granting ticket and the server's target principal name are valid, the TGS issues a service ticket to the client. The principal name recorded in the database directory can be specified as name/instance@REALM. (This is in addition to DOMAIN\userID and userID@xxx.xxx.xxx.com formats accepted on Windows.)
3. The client sends this service ticket to the server using the communication channel (which can be, as an example, TCP/IP).
4. The server validates the client's server ticket. If the client's service ticket is valid, then the authentication is completed.

It is possible to catalog the databases on the client machine and explicitly specify the Kerberos authentication type with the server's target principal name. In this way, the first phase of the connection can be bypassed.

If a user ID and a password are specified, the client will request the ticket-granting ticket for that user account and use it for authentication.

KRB_SERVER_ENCRYPT

Specifies that the server accepts KERBEROS authentication or encrypted SERVER authentication schemes. If the client authentication is KERBEROS, the client is authenticated using the Kerberos security system. If the client authentication is SERVER_ENCRYPT, the client is authenticated using a user ID and encryption password. If the client authentication is not specified, then the client will use Kerberos if available, otherwise it will use password encryption. For other client authentication types, an authentication error is returned. The authentication type of the client cannot be specified as KRB_SERVER_ENCRYPT

Note: The Kerberos authentication types are supported on clients and servers running on specific operating systems. For Windows operating systems, both client and server machines must either belong to the same Windows domain or belong to trusted domains. This authentication type should be used when the server supports Kerberos and some, but not all, of the client machines support Kerberos authentication.

GSSPLUGIN

Specifies that the server uses a GSS-API plug-in to perform authentication. If the client authentication is not specified, the server returns a list of server-supported plug-ins, including any Kerberos plug-in that is listed in the **srvcon_gssplugin_list** database manager configuration parameter, to the client. The client selects the first plug-in found in the client plug-in directory from the list. If the client does not support any plug-in in the list, the client is authenticated using the Kerberos authentication scheme (if it is returned). If the client authentication is the GSSPLUGIN authentication scheme, the client is authenticated using the first supported plug-in in the list.

GSS_SERVER_ENCRYPT

Specifies that the server accepts plug-in authentication or encrypted server authentication schemes. If client authentication occurs through a plug-in, the client is authenticated using the first client-supported plug-in in the list of server-supported plug-ins.

If the client authentication is not specified and an implicit connect is being performed (that is, the client does not supply a user ID and password when making the connection), the server returns a list of server-supported plug-ins, the Kerberos authentication scheme (if one of the plug-ins in the list is Kerberos-based), and the encrypted server authentication scheme. The client is authenticated using the first supported plug-in found in the client plug-in directory. If the client does not support any of the plug-ins that are in the list, the client is authenticated using the Kerberos authentication scheme. If the client does not support the Kerberos authentication scheme, the client is authenticated using the encrypted server authentication scheme, and the connection will fail because of a missing password. A client supports the Kerberos authentication scheme if a DB2 supplied Kerberos plug-in exists for the operating system, or a Kerberos-based plug-in is specified for the **srvcon_gssplugin_list** database manager configuration parameter.

If the client authentication is not specified and an explicit connection is being performed (that is, both the user ID and password are supplied), the authentication type is equivalent to SERVER_ENCRYPT. In this case, the choice of the encryption algorithm used to encrypt the user ID and password depends on the setting of the **alternate_auth_enc** database manager configuration parameter.

Note:

1. Do not inadvertently lock yourself out of your instance when you are changing the authentication information, since access to the configuration file itself is protected by information in the configuration file. The following database manager configuration file parameters control access to the instance:
   - **authentication** *
   - **sysadm_group** *
   - **trust_allclnts**
   - **trust_clntauth**
   - **sysctrl_group**
   - **sysmaint_group**

   * Indicates the two most important parameters.

   There are some things that can be done to ensure this does not happen: If you do accidentally lock yourself out of the DB2 database system, you have a fail-safe option available on all platforms that will allow you to override the usual DB2 database security checks to update the database manager configuration file using a highly privileged local operating system security user. This user *always* has the privilege to update the database manager configuration file and thereby correct the problem. However, this security bypass is restricted to a local update of the database manager configuration file. You cannot use a fail-safe user remotely or for any other DB2 database command. This special user is identified as follows:
   - UNIX platforms: the instance owner
   - Windows platform: someone belonging to the local “Administrators” group
   - Other platforms: there is no local security on the other platforms, so all users pass local security checks anyway

Chapter 53. Authorization, privileges, and object ownership

Users (identified by an authorization ID) can successfully execute operations only if they have the authority to perform the specified function. To create a table, a user must be authorized to create tables; to alter a table, a user must be authorized to alter the table; and so forth.

The database manager requires that each user be specifically authorized to use each database function needed to perform a specific task. A user can acquire the necessary authorization through a grant of that authorization to their user ID or through membership in a role or a group that holds that authorization.

There are three forms of authorization, *administrative authority*, *privileges*, and *LBAC credentials*. In addition, ownership of objects brings with it a degree of authorization on the objects created. These forms of authorization are discussed in the following section.

Administrative authority

The person or persons holding administrative authority are charged with the task of controlling the database manager and are responsible for the safety and integrity of the data.

System-level authorization

The system-level authorities provide varying degrees of control over instance-level functions:

- SYSADM (system administrator) authority

  The SYSADM (system administrator) authority provides control over all the resources created and maintained by the database manager. The system administrator possesses all the authorities of SYSCTRL, SYSMAINT, and SYSMON authority. The user who has SYSADM authority is responsible both for controlling the database manager, and for ensuring the safety and integrity of the data.
- SYSCTRL authority

  The SYSCTRL authority provides control over operations that affect system resources. For example, a user with SYSCTRL authority can create, update, start, stop, or drop a database. This user can also start or stop an instance, but cannot access table data. Users with SYSCTRL authority also have SYSMON authority.
- SYSMAINT authority

  The SYSMAINT authority provides the authority required to perform maintenance operations on all databases associated with an instance. A user with SYSMAINT authority can update the database configuration, backup a database or table space, restore an existing database, and monitor a database. Like SYSCTRL, SYSMAINT does not provide access to table data. Users with SYSMAINT authority also have SYSMON authority.
- SYSMON (system monitor) authority

  The SYSMON (system monitor) authority provides the authority required to use the database system monitor.

Database-level authorization

The database level authorities provide control within the database:

- DBADM (database administrator)

  The DBADM authority level provides administrative authority over a single database. This database administrator possesses the privileges required to create objects and issue database commands.

  The DBADM authority can be granted only by a user with SECADM authority. The DBADM authority cannot be granted to PUBLIC.
- SECADM (security administrator)

  The SECADM authority level provides administrative authority for security over a single database. The security administrator authority possesses the ability to manage database security objects (database roles, audit policies, trusted contexts, security label components, and security labels) and grant and revoke all database privileges and authorities. A user with SECADM authority can transfer the ownership of objects that they do not own. They can also use the AUDIT statement to associate an audit policy with a particular database or database object at the server.

  The SECADM authority has no inherent privilege to access data stored in tables. It can only be granted by a user with SECADM authority. The SECADM authority cannot be granted to PUBLIC.
- SQLADM (SQL administrator)

  The SQLADM authority level provides administrative authority to monitor and tune SQL statements within a single database. It can be granted by a user with ACCESSCTRL or SECADM authority.
- WLMADM (workload management administrator)

  The WLMADM authority provides administrative authority to manage workload management objects, such as service classes, work action sets, work class sets, and workloads. It can be granted by a user with ACCESSCTRL or SECADM authority.
- EXPLAIN (explain authority)

  The EXPLAIN authority level provides administrative authority to explain query plans without gaining access to data. It can only be granted by a user with ACCESSCTRL or SECADM authority.
- ACCESSCTRL (access control authority)

  The ACCESSCTRL authority level provides administrative authority to issue the following GRANT (and REVOKE) statements.

  – GRANT (Database Authorities)

    ACCESSCTRL authority does not give the holder the ability to grant ACCESSCTRL, DATAACCESS, DBADM, or SECADM authority. Only a user who has SECADM authority can grant these authorities.

  – GRANT (Global Variable Privileges)

  – GRANT (Index Privileges)

  – GRANT (Module Privileges)

  – GRANT (Package Privileges)

  – GRANT (Routine Privileges)

  – GRANT (Schema Privileges)

  – GRANT (Sequence Privileges)

  – GRANT (Server Privileges)

  – GRANT (Table, View, or Nickname Privileges)

  – GRANT (Table Space Privileges)

- GRANT (Workload Privileges)
- GRANT (XSR Object Privileges)

ACCESSCTRL authority can only be granted by a user with SECADM authority. The ACCESSCTRL authority cannot be granted to PUBLIC.

- DATAACCESS (data access authority)

  The DATAACCESS authority level provides the following privileges and authorities.

  - LOAD authority
  - SELECT, INSERT, UPDATE, DELETE privilege on tables, views, nicknames, and materialized query tables
  - EXECUTE privilege on packages
  - EXECUTE privilege on modules
  - EXECUTE privilege on routines

    Except on the audit routines: AUDIT_ARCHIVE, AUDIT_LIST_LOGS, AUDIT_DELIM_EXTRACT.
  - READ privilege on all global variables and WRITE privilege on all global variables except variables which are read-only
  - USAGE privilege on all XSR objects
  - USAGE privilege on all sequences

  It can be granted only by a user who holds SECADM authority. The DATAACCESS authority cannot be granted to PUBLIC.
- Database authorities (non-administrative)

  To perform activities such as creating a table or a routine, or for loading data into a table, specific database authorities are required. For example, the LOAD database authority is required for use of the `load` utility to load data into tables (a user must also have INSERT privilege on the table).

Privileges

A privilege is a permission to perform an action or a task. Authorized users can create objects, have access to objects they own, and can pass on privileges on their own objects to other users by using the GRANT statement.

Privileges may be granted to individual users, to groups, or to PUBLIC. PUBLIC is a special group that consists of all users, including future users. Users that are members of a group will indirectly take advantage of the privileges granted to the group, where groups are supported.

The CONTROL privilege: Possessing the CONTROL privilege on an object allows a user to access that database object, and to grant and revoke privileges to or from other users on that object.

Note: The CONTROL privilege only apples to tables, views, nicknames, indexes, and packages.

If a different user requires the CONTROL privilege to that object, a user with SECADM or ACCESSCTRL authority could grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked from the object owner, however, the object owner can be changed by using the TRANSFER OWNERSHIP statement.

Individual privileges: Individual privileges can be granted to allow a user to carry out specific tasks on specific objects. Users with the administrative authorities ACCESSCTRL or SECADM, or with the CONTROL privilege, can grant and revoke privileges to and from users.

Individual privileges and database authorities allow a specific function, but do not include the right to grant the same privileges or authorities to other users. The right to grant table, view, schema, package, routine, and sequence privileges to others can be extended to other users through the WITH GRANT OPTION on the GRANT statement. However, the WITH GRANT OPTION does not allow the person granting the privilege to revoke the privilege once granted. You must have SECADM authority, ACCESSCTRL authority, or the CONTROL privilege to revoke the privilege.

Privileges on objects in a package or routine: When a user has the privilege to execute a package or routine, they do not necessarily require specific privileges on the objects used in the package or routine. If the package or routine contains static SQL or XQuery statements, the privileges of the owner of the package are used for those statements. If the package or routine contains dynamic SQL or XQuery statements, the authorization ID used for privilege checking depends on the setting of the **DYNAMICRULES BIND** option of the package issuing the dynamic query statements, and whether those statements are issued when the package is being used in the context of a routine (except on the audit routines: AUDIT_ARCHIVE, AUDIT_LIST_LOGS, AUDIT_DELIM_EXTRACT).

A user or group can be authorized for any combination of individual privileges or authorities. When a privilege is associated with an object, that object must exist. For example, a user cannot be given the SELECT privilege on a table unless that table has previously been created.

Note: Care must be taken when an authorization name representing a user or a group is granted authorities and privileges and there is no user, or group created with that name. At some later time, a user or a group can be created with that name and automatically receive all of the authorities and privileges associated with that authorization name.

The REVOKE statement is used to revoke previously granted privileges. The revoking of a privilege from an authorization name revokes the privilege granted by all authorization names.

Revoking a privilege from an authorization name does not revoke that same privilege from any other authorization names that were granted the privilege by that authorization name. For example, assume that CLAIRE grants SELECT WITH GRANT OPTION to RICK, then RICK grants SELECT to BOBBY and CHRIS. If CLAIRE revokes the SELECT privilege from RICK, BOBBY and CHRIS still retain the SELECT privilege.

LBAC credentials

Label-based access control (LBAC) lets the security administrator decide exactly who has write access and who has read access to individual rows and individual columns. The security administrator configures the LBAC system by creating security policies. A security policy describes the criteria used to decide who has access to what data. Only one security policy can be used to protect any one table but different tables can be protected by different security policies.

After creating a security policy, the security administrator creates database objects, called security labels and exemptions that are part of that policy. A security label describes a certain set of security criteria. An exemption allows a rule for comparing security labels not to be enforced for the user who holds the exemption, when they access data protected by that security policy.

Once created, a security label can be associated with individual columns and rows in a table to protect the data held there. Data that is protected by a security label is called protected data. A security administrator allows users access to protected data by granting them security labels. When a user tries to access protected data, that user's security label is compared to the security label protecting the data. The protecting label blocks some security labels and does not block others.

Object ownership

When an object is created, one authorization ID is assigned *ownership* of the object. Ownership means the user is authorized to reference the object in any applicable SQL or XQuery statement.

When an object is created within a schema, the authorization ID of the statement must have the required privilege to create objects in the implicitly or explicitly specified schema. That is, the authorization name must either be the owner of the schema, or possess the CREATEIN privilege on the schema.

Note: This requirement is not applicable when creating table spaces, buffer pools or database partition groups. These objects are not created in schemas.

When an object is created, the authorization ID of the statement is the definer of that object and by default becomes the owner of the object after it is created.

Note: One exception exists. If the AUTHORIZATION option is specified for the CREATE SCHEMA statement, any other object that is created as part of the CREATE SCHEMA operation is owned by the authorization ID specified by the AUTHORIZATION option. Any objects that are created in the schema after the initial CREATE SCHEMA operation, however, are owned by the authorization ID associated with the specific CREATE statement.

For example, the statement `CREATE SCHEMA SCOTTSTUFF AUTHORIZATION SCOTT CREATE TABLE T1 (C1 INT)` creates the schema `SCOTTSTUFF` and the table `SCOTTSTUFF.T1`, which are both owned by SCOTT. Assume that the user BOBBY is granted the CREATEIN privilege on the `SCOTTSTUFF` schema and creates an index on the `SCOTTSTUFF.T1` table. Because the index is created after the schema, BOBBY owns the index on `SCOTTSTUFF.T1`.

Privileges are assigned to the object owner based on the type of object being created:

- The CONTROL privilege is implicitly granted on newly created tables, indexes, and packages. This privilege allows the object creator to access the database object, and to grant and revoke privileges to or from other users on that object. If a different user requires the CONTROL privilege to that object, a user with ACCESSCTRL or SECADM authority must grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked by the object owner.
- The CONTROL privilege is implicitly granted on newly created views if the object owner has the CONTROL privilege on all the tables, views, and nicknames referenced by the view definition.

- Other objects like triggers, routines, sequences, table spaces, and buffer pools do not have a CONTROL privilege associated with them. The object owner does, however, automatically receive each of the privileges associated with the object and those privileges are with the WITH GRANT OPTION, where supported. Therefore the object owner can provide these privileges to other users by using the GRANT statement. For example, if USER1 creates a table space, USER1 automatically has the USEAUTH privilege with the WITH GRANT OPTION on this table space and can grant the USEAUTH privilege to other users. In addition, the object owner can alter, add a comment on, or drop the object. These authorizations are implicit for the object owner and cannot be revoked.

Certain privileges on the object, such as altering a table, can be granted by the owner, and can be revoked from the owner by a user who has ACCESSCTRL or SECADM authority. Certain privileges on the object, such as commenting on a table, cannot be granted by the owner and cannot be revoked from the owner. Use the TRANSFER OWNERSHIP statement to move these privileges to another user. When an object is created, the authorization ID of the statement is the definer of that object and by default becomes the owner of the object after it is created. However, when you use the **BIND** command to create a package and you specify the **OWNER** *authorization id* option, the owner of objects created by the static SQL statements in the package is the value of *authorization id*. In addition, if the AUTHORIZATION clause is specified on a CREATE SCHEMA statement, the authorization name specified after the AUTHORIZATION keyword is the owner of the schema.

A security administrator or the object owner can use the TRANSFER OWNERSHIP statement to change the ownership of a database object. An administrator can therefore create an object on behalf of an authorization ID, by creating the object using the authorization ID as the qualifier, and then using the TRANSFER OWNERSHIP statement to transfer the ownership that the administrator has on the object to the authorization ID.

Chapter 54. Default privileges granted on creating a database

When you create a database, default database level authorities and default object level privileges are granted to you within that database.

The authorities and privileges that you are granted are listed according to the system catalog views where they are recorded:

1. SYSCAT.DBAUTH
   - The database creator is granted the following authorities:
     - ACCESSCTRL
     - DATAACCESS
     - DBADM
     - SECADM
   - In a non-restrictive database, the special group PUBLIC is granted the following authorities:
     - CREATETAB
     - BINDADD
     - CONNECT
     - IMPLICIT_SCHEMA
2. SYSCAT.TABAUTH

   In a non-restrictive database, the special group PUBLIC is granted the following privileges:
   - SELECT on all SYSCAT and SYSIBM tables
   - SELECT and UPDATE on all SYSSTAT tables
   - SELECT on the following views in schema SYSIBMADM:
     - ALL_*
     - USER_*
     - ROLE_*
     - SESSION_*
     - DICTIONARY
     - TAB
3. SYSCAT.ROUTINEAUTH

   In a non-restrictive database, the special group PUBLIC is granted the following privileges:
   - EXECUTE with GRANT on all procedures in schema SQLJ
   - EXECUTE with GRANT on all functions and procedures in schema SYSFUN
   - EXECUTE with GRANT on all functions and procedures in schema SYSPROC (except audit routines)
   - EXECUTE on all table functions in schema SYSIBM
   - EXECUTE on all other procedures in schema SYSIBM
4. SYSCAT.MODULEAUTH

   In a non-restrictive database, the special group PUBLIC is granted the following privileges:
   - EXECUTE on the following modules in schema SYSIBMADM:
     - DBMS_DDL

   - DBMS_JOB
   - DBMS_LOB
   - DBMS_OUTPUT
   - DBMS_SQL
   - DBMS_STANDARD
   - DBMS_UTILITY
5. SYSCAT.PACKAGEAUTH
   - The database creator is granted the following privileges:
     - CONTROL on all packages created in the NULLID schema
     - BIND with GRANT on all packages created in the NULLID schema
     - EXECUTE with GRANT on all packages created in the NULLID schema
   - In a non-restrictive database, the special group PUBLIC is granted the following privileges:
     - BIND on all packages created in the NULLID schema
     - EXECUTE on all packages created in the NULLID schema
6. SYSCAT.SCHEMAAUTH

   In a non-restrictive database, the special group PUBLIC is granted the following privileges:
   - CREATEIN on schema SQLJ
   - CREATEIN on schema NULLID
7. SYSCAT.TBSPACEAUTH

   In a non-restrictive database, the special group PUBLIC is granted the USE privilege on table space USERSPACE1.
8. SYSCAT.WORKLOADAUTH

   In a non-restrictive database, the special group PUBLIC is granted the USAGE privilege on SYSDEFAULTUSERWORKLOAD.
9. SYSCAT.VARIABLEAUTH

   In a non-restrictive database, the special group PUBLIC is granted the READ privilege on schema global variables in the SYSIBM schema, execpt for the following variables:
   - SYSIBM.CLIENT_ORIGUSERID
   - SYSIBM.CLIENT_USRSECTOKEN

A non-restrictive database is a database created without the RESTRICTIVE option on the CREATE DATABASE command.

Chapter 55. Granting privileges

To grant privileges on most database objects, you must have ACCESSCTRL authority, SECADM authority, or CONTROL privilege on that object; or, you must hold the privilege WITH GRANT OPTION. Additionally, users with SYSADM or SYSCTRL authority can grant table space privileges. You can grant privileges only on existing objects.

About this task

To grant CONTROL privilege to someone else, you must have ACCESSCTRL or SECADM authority. To grant ACCESSCTRL, DATAACCESS, DBADM or SECADM authority, you must have SECADM authority.

The GRANT statement allows an authorized user to grant privileges. A privilege can be granted to one or more authorization names in one statement; or to PUBLIC, which makes the privileges available to all users. Note that an authorization name can be either an individual user or a group.

On operating systems where users and groups exist with the same name, you should specify whether you are granting the privilege to the user or group. Both the GRANT and REVOKE statements support the keywords USER, GROUP, and ROLE. If these optional keywords are not used, the database manager checks the operating system security facility to determine whether the authorization name identifies a user or a group; it also checks whether an authorization ID of type role with the same name exists. If the database manager cannot determine whether the authorization name refers to a user, a group, or a role, an error is returned. The following example grants SELECT privileges on the EMPLOYEE table to the user HERON:

```
GRANT SELECT
 ON EMPLOYEE TO USER HERON
```

The following example grants SELECT privileges on the EMPLOYEE table to the group HERON:

```
GRANT SELECT
   ON EMPLOYEE TO GROUP HERON
```

Chapter 56. Revoking privileges

The REVOKE statement allows authorized users to revoke privileges previously granted to other users.

About this task

To revoke privileges on database objects, you must have ACCESSCTRL authority, SECADM authority, or CONTROL privilege on that object. Table space privileges can also be revoked by users with SYSADM and SYSCTRL authority. Note that holding a privilege WITH GRANT OPTION is not sufficient to revoke that privilege. To revoke CONTROL privilege from another user, you must have ACCESSCTRL, or SECADM authority. To revoke ACCESSCTRL, DATAACCESS, DBADM or SECADM authority, you must have SECADM authority. Table space privileges can be revoked only by a user who holds SYSADM, or SYSCTRL authority. Privileges can only be revoked on existing objects.

Note: A user without ACCESSCTRL authority, SECADM authority, or CONTROL privilege is not able to revoke a privilege that they granted through their use of the WITH GRANT OPTION. Also, there is no cascade on the revoke to those who have received privileges granted by the person being revoked.
If an explicitly granted table (or view) privilege is revoked from a user with DBADM authority, privileges will not be revoked from other views defined on that table. This is because the view privileges are available through the DBADM authority and are not dependent on explicit privileges on the underlying tables.

If a privilege has been granted to a user, a group, or a role with the same name, you must specify the GROUP, USER, or ROLE keyword when revoking the privilege. The following example revokes the SELECT privilege on the EMPLOYEE table from the user HERON:

```
REVOKE SELECT
   ON EMPLOYEE FROM USER HERON
```

The following example revokes the SELECT privilege on the EMPLOYEE table from the group HERON:

```
REVOKE SELECT
   ON EMPLOYEE FROM GROUP HERON
```

Note that revoking a privilege from a group may not revoke it from all members of that group. If an individual name has been directly granted a privilege, it will keep it until that privilege is directly revoked.

If a table privilege is revoked from a user, privileges are also revoked on any view created by that user which depends on the revoked table privilege. However, only the privileges implicitly granted by the system are revoked. If a privilege on the view was granted directly by another user, the privilege is still held.

If a table privilege is revoked from a user, privileges are also revoked on any view created by that user which depends on the revoked table privilege. However, only the privileges implicitly granted by the system are revoked. If a privilege on the view was granted directly by another user, the privilege is still held.

You may have a situation where you want to GRANT a privilege to a group and then REVOKE the privilege from just one member of the group. There are only a couple of ways to do that without receiving the error message SQL0556N:

- You can remove the member from the group; or, create a new group with fewer members and GRANT the privilege to the new group.
- You can REVOKE the privilege from the group and then GRANT it to individual users (authorization IDs).

Note: When CONTROL privilege is revoked from a user on a table or a view, the user continues to have the ability to grant privileges to others. When given CONTROL privilege, the user also receives all other privileges WITH GRANT OPTION. Once CONTROL is revoked, all of the other privileges remain WITH GRANT OPTION until they are explicitly revoked.

All packages that are dependent on revoked privileges are marked invalid, but can be validated if rebound by a user with appropriate authority. Packages can also be rebuilt if the privileges are subsequently granted again to the binder of the application; running the application will trigger a successful implicit rebind. If privileges are revoked from PUBLIC, all packages bound by users having only been able to bind based on PUBLIC privileges are invalidated. If DBADM authority is revoked from a user, all packages bound by that user are invalidated including those associated with database utilities. Attempting to use a package that has been marked invalid causes the system to attempt to rebind the package. If this rebind attempt fails, an error occurs (SQLCODE -727). In this case, the packages must be explicitly rebound by a user with:

- Authority to rebind the packages
- Appropriate authority for the objects used within the packages

These packages should be rebound at the time the privileges are revoked.

If you define a trigger or SQL function based on one or more privileges and you lose one or more of these privileges, the trigger or SQL function cannot be used.

Chapter 57. Controlling access to data with views

A view provides a means of controlling access or extending privileges to a table.

Using a view allows the following kinds of control over access to a table:

- Access only to designated columns of the table.

  For users and application programs that require access only to specific columns of a table, an authorized user can create a view to limit the columns addressed only to those required.
- Access only to a subset of the rows of the table.

  By specifying a WHERE clause in the subquery of a view definition, an authorized user can limit the rows addressed through a view.
- Access only to a subset of the rows or columns in data source tables or views. If you are accessing data sources through nicknames, you can create local DB2 database views that reference nicknames. These views can reference nicknames from one or many data sources.

  Note: Because you can create a view that contains nickname references for more than one data source, your users can access data in multiple data sources from one view. These views are called *multi-location views*. Such views are useful when joining information in columns of sensitive tables across a distributed environment or when individual users lack the privileges needed at data sources for specific objects.

To create a view, a user must have DATAACCESS authority, or CONTROL or SELECT privilege for each table, view, or nickname referenced in the view definition. The user must also be able to create an object in the schema specified for the view. That is, DBADM authority, CREATEIN privilege for an existing schema, or IMPLICIT_SCHEMA authority on the database if the schema does not already exist.

If you are creating views that reference nicknames, you do not need additional authority on the data source objects (tables and views) referenced by nicknames in the view; however, users of the view must have SELECT authority or the equivalent authorization level for the underlying data source objects when they access the view.

If your users do not have the proper authority at the data source for underlying objects (tables and views), you can:

1. Create a data source view over those columns in the data source table that are OK for the user to access
2. Grant the SELECT privilege on this view to users
3. Create a nickname to reference the view

Users can then access the columns by issuing a SELECT statement that references the new nickname.

The following scenario provides a more detailed example of how views can be used to restrict access to information.

Many people might require access to information in the STAFF table, for different reasons. For example:

- The personnel department needs to be able to update and look at the entire table.

  This requirement can be easily met by granting SELECT and UPDATE privileges on the STAFF table to the group PERSONNL:

  ```
  GRANT SELECT,UPDATE ON TABLE STAFF TO GROUP PERSONNL
  ```
- Individual department managers need to look at the salary information for their employees.

  This requirement can be met by creating a view for each department manager. For example, the following view can be created for the manager of department number 51:

  ```
  CREATE VIEW EMP051 AS
     SELECT NAME,SALARY,JOB FROM STAFF
     WHERE DEPT=51
  GRANT SELECT ON TABLE EMP051 TO JANE
  ```

  The manager with the authorization name JANE would query the EMP051 view just like the STAFF table. When accessing the EMP051 view of the STAFF table, this manager views the following information:

NAME	SALARY	JOB
Fraye	45150.0	Mgr
Williams	37156.5	Sales
Smith	35654.5	Sales
Lundquist	26369.8	Clerk
Wheeler	22460.0	Clerk

- All users need to be able to locate other employees. This requirement can be met by creating a view on the NAME column of the STAFF table and the LOCATION column of the ORG table, and by joining the two tables on their corresponding DEPT and DEPTNUMB columns:

  ```
  CREATE VIEW EMPLOCS AS
     SELECT NAME, LOCATION FROM STAFF, ORG
     WHERE STAFF.DEPT=ORG.DEPTNUMB
  GRANT SELECT ON TABLE EMPLOCS TO PUBLIC
  ```

  Users who access the employee location view will see the following information:

NAME	LOCATION
Molinare	New York
Lu	New York
Daniels	New York
Jones	New York
Hanes	Boston
Rothman	Boston
Ngan	Boston
Kermisch	Boston
Sanders	Washington
Pernal	Washington
James	Washington
Sneider	Washington
Marenghi	Atlanta

NAME	LOCATION
O'Brien	Atlanta
Quigley	Atlanta
Naughton	Atlanta
Abrahams	Atlanta
Koonitz	Chicago
Plotz	Chicago
Yamaguchi	Chicago
Scoutten	Chicago
Fraye	Dallas
Williams	Dallas
Smith	Dallas
Lundquist	Dallas
Wheeler	Dallas
Lea	San Francisco
Wilson	San Francisco
Graham	San Francisco
Gonzales	San Francisco
Burke	San Francisco
Quill	Denver
Davis	Denver
Edwards	Denver
Gafney	Denver

Chapter 58. Roles

Roles simplify the administration and management of privileges by offering an equivalent capability as groups but without the same restrictions.

A role is a database object that groups together one or more privileges and can be assigned to users, groups, PUBLIC, or other roles by using a GRANT statement, or can be assigned to a trusted context by using a CREATE TRUSTED CONTEXT or ALTER TRUSTED CONTEXT statement. A role can be specified for the SESSION_USER ROLE connection attribute in a workload definition.

Roles provide several advantages that make it easier to manage privileges in a database system:

- Security administrators can control access to their databases in a way that mirrors the structure of their organizations (they can create roles in the database that map directly to the job functions in their organizations).
- Users are granted membership in the roles that reflect their job responsibilities. As their job responsibilities change, their membership in roles can be easily granted and revoked.
- The assignment of privileges is simplified. Instead of granting the same set of privileges to each individual user in a particular job function, the administrator can grant this set of privileges to a role representing that job function and then grant that role to each user in that job function.
- A role's privileges can be updated and all users who have been granted that role receive the update; the administrator does not need to update the privileges for every user on an individual basis.
- The privileges and authorities granted to roles are always used when you create views, triggers, materialized query tables (MQTs), static SQL and SQL routines, whereas privileges and authorities granted to groups (directly or indirectly) are not used.

  This is because the DB2 database system cannot determine when membership in a group changes, as the group is managed by third-party software (for example, the operating system or an LDAP directory). Because roles are managed inside the database, the DB2 database system can determine when authorization changes and act accordingly. Roles granted to groups are not considered, due to the same reason groups are not considered.
- All the roles assigned to a user are enabled when that user establishes a connection, so all privileges and authorities granted to roles are taken into account when a user connects. Roles cannot be explicitly enabled or disabled.
- The security administrator can delegate management of a role to others.

All DB2 privileges and authorities that can be granted within a database can be granted to a role. For example, a role can be granted any of the following authorities and privileges:

- DBADM, SECADM, DATAACCESS, ACCESSCTRL, SQLADM, WLMADM, LOAD, and IMPLICIT_SCHEMA database authorities
- CONNECT, CREATETAB, CREATE_NOT_FENCED, BINDADD, CREATE_EXTERNAL_ROUTINE, or QUIESCE_CONNECT database authorities
- Any database object privilege (including CONTROL)

A user's roles are automatically enabled and considered for authorization when a user connects to a database; you do not need to activate a role by using the SET ROLE statement. For example, when you create a view, a materialized query table (MQT), a trigger, a package, or an SQL routine, the privileges that you gain through roles apply. However, privileges that you gain through roles granted to groups of which you are a member do not apply.

A role does not have an owner. The security administrator can use the WITH ADMIN OPTION clause of the GRANT statement to delegate management of the role to another user, so that the other user can control the role membership.

Restrictions

There are a few restrictions in the use of roles:

- A role cannot own database objects.
- Permissions and roles granted to groups are not considered when you create the following database objects:
  - Packages containing static SQL
  - Views
  - Materialized query tables (MQT)
  - Triggers
  - SQL Routines

  Only roles granted to the user creating the object or to PUBLIC, directly or indirectly (such as through a role hierarchy), are considered when creating these objects.

Roles compared to groups

Privileges and authorities granted to groups are not considered when creating views, materialized query tables (MQTs), SQL routines, triggers, and packages containing static SQL. Avoid this restriction by using roles instead of groups.

Roles allow users to create database objects using their privileges acquired through roles, which are controlled by the DB2 database system. Groups and users are controlled externally from the DB2 database system, for example, by an operating system or an LDAP server.

Example of replacing the use of groups with roles

This example shows how you can replace groups by using roles.

Assume that there are three groups, DEVELOPER_G, TESTER_G and SALES_G. The users BOB, ALICE, and TOM are members of these groups, as shown in the following table:

Table 136. Example groups and users

Group	Users belonging to this group
DEVELOPER_G	BOB
TESTER_G	ALICE, TOM
SALES_G	ALICE, BOB

1. The security administrator creates the roles DEVELOPER, TESTER, and SALES to be used instead of the groups.

```
CREATE ROLE DEVELOPER
CREATE ROLE TESTER
CREATE ROLE SALES
```

2. The security administrator grants membership in these roles to users (setting the membership of users in groups was the responsibility of the system administrator):

   ```
   GRANT  ROLE DEVELOPER TO USER BOB
   GRANT  ROLE TESTER TO USER ALICE, USER TOM
   GRANT  ROLE SALES TO USER BOB, USER ALICE
   ```

3. The database administrator can grant to the roles similar privileges or authorities as were held by the groups, for example:

   ```
   GRANT privilege ON object TO ROLE DEVELOPER
   ```

   The database administrator can then revoke these privileges from the groups, as well as ask the system administrator to remove the groups from the system.

Example of creating a trigger using privileges acquired through a role

This example shows that user BOB can successfully create a trigger, TRG1, when he holds the necessary privilege through the role DEVELOPER.

1. First, user ALICE creates the table, WORKITEM:

   ```
   CREATE TABLE WORKITEM (x int)
   ```

2. Then, the privilege to alter ALICE's table is granted to role DEVELOPER by the database administrator.

   ```
   GRANT ALTER ON ALICE.WORKITEM TO ROLE DEVELOPER
   ```

3. User BOB successfully creates the trigger, TRG1, because he is a member of the role, DEVELOPER.

   ```
   CREATE TRIGGER TRG1 AFTER DELETE ON ALICE.WORKITEM
             FOR EACH STATEMENT MODE DB2SQL INSERT INTO ALICE.WORKITEM VALUES (1)
   ```

Chapter 59. Trusted contexts and trusted connections

A trusted context is a database object that defines a trust relationship for a connection between the database and an external entity such as an application server.

The trust relationship is based upon the following set of attributes:

- System authorization ID: Represents the user that establishes a database connection
- IP address (or domain name): Represents the host from which a database connection is established
- Data stream encryption: Represents the encryption setting (if any) for the data communication between the database server and the database client

When a user establishes a database connection, the DB2 database system checks whether the connection matches the definition of a trusted context object in the database. When a match occurs, the database connection is said to be trusted.

A trusted connection cannot be established if the connection is to a local database using inter-process communication (IPC).

A trusted connection allows the initiator of this trusted connection to acquire additional capabilities that may not be available outside the scope of the trusted connection. The additional capabilities vary depending on whether the trusted connection is explicit or implicit.

The initiator of an explicit trusted connection has the ability to:

- Switch the current user ID on the connection to a different user ID with or without authentication
- Acquire additional privileges via the role inheritance feature of trusted contexts

An implicit trusted connection is a trusted connection that is not explicitly requested; the implicit trusted connection results from a normal connection request rather than an explicit trusted connection request. No application code changes are needed to obtain an implicit connection. Also, whether you obtain an implicit trusted connection or not has no effect on the connect return code (when you request an explicit trusted connection, the connect return code indicates whether the request succeeds or not). The initiator of an implicit trusted connection can only acquire additional privileges via the role inheritance feature of trusted contexts; they cannot switch the user ID.

How using trusted contexts enhances security

The three-tiered application model extends the standard two-tiered client and server model by placing a middle tier between the client application and the database server. It has gained great popularity in recent years particularly with the emergence of web-based technologies and the Java 2 Enterprise Edition (J2EE) platform. An example of a software product that supports the three-tier application model is IBM WebSphere® Application Server (WAS).

In a three-tiered application model, the middle tier is responsible for authenticating the users running the client applications and for managing the interactions with the database server. Traditionally, all the interactions with the database server occur through a database connection established by the middle tier using a combination of a user ID and a credential that identify that middle tier to the database server. This means that the database server uses the database privileges associated with the middle tier's user ID for all authorization checking and auditing that must occur for any database access, including access performed by the middle tier on behalf of a user.

While the three-tiered application model has many benefits, having all interactions with the database server (for example, a user request) occur under the middle tier's authorization ID raises several security concerns, which can be summarized as follows:

- Loss of user identity

  Some enterprises prefer to know the identity of the actual user accessing the database for access control purposes.
- Diminished user accountability

  Accountability through auditing is a basic principle in database security. Not knowing the user's identity makes it difficult to distinguish the transactions performed by the middle tier for its own purpose from those performed by the middle tier on behalf of a user.
- Over granting of privileges to the middle tier's authorization ID

  The middle tier's authorization ID must have all the privileges necessary to execute all the requests from all the users. This has the security issue of enabling users who do not need access to certain information to obtain access anyway.
- Weakened security

  In addition to the privilege issue raised in the previous point, the current approach requires that the authorization ID used by the middle tier to connect must be granted privileges on all resources that might be accessed by user requests. If that middle-tier authorization ID is ever compromised, then all those resources will be exposed.
- "Spill over" between users of the same connection

  Changes by a previous user can affect the current user.

Clearly, there is a need for a mechanism whereby the actual user's identity and database privileges are used for database requests performed by the middle tier on behalf of that user. The most straightforward approach of achieving this goal would be for the middle-tier to establish a new connection using the user's ID and password, and then direct the user's requests through that connection. Although simple, this approach suffers from several drawbacks which include the following:

- Inapplicability for certain middle tiers. Many middle-tier servers do not have the user authentication credentials needed to establish a connection.
- Performance overhead. There is an obvious performance overhead associated with creating a new physical connection and re-authenticating the user at the database server.
- Maintenance overhead. In situations where you are not using a centralized security set up or are not using single sign-on, there is maintenance overhead in having two user definitions (one on the middle tier and one at the server). This requires changing passwords at different places.

The trusted contexts capability addresses this problem. The security administrator can create a trusted context object in the database that defines a trust relationship between the database and the middle-tier. The middle-tier can then establish an explicit trusted connection to the database, which gives the middle tier the ability to switch the current user ID on the connection to a different user ID, with or without authentication. In addition to solving the end-user identity assertion problem, trusted contexts offer another advantage. This is the ability to control when a privilege is made available to a database user. The lack of control on when privileges are available to a user can weaken overall security. For example, privileges may be used for purposes other than they were originally intended. The security administrator can assign one or more privileges to a role and assign that role to a trusted context object. Only trusted database connections (explicit or implicit) that match the definition of that trusted context can take advantage of the privileges associated with that role.

Enhancing performance

When you use trusted connections, you can maximize performance because of the following advantages:

- No new connection is established when the current user ID of the connection is switched.
- If the trusted context definition does not require authentication of the user ID to switch to, then the overhead associated with authenticating a new user at the database server is not incurred.

Example of creating a trusted context

Suppose that the security administrator creates the following trusted context object:

```
CREATE TRUSTED CONTEXT CTX1
  BASED UPON CONNECTION USING SYSTEM AUTHID USER2
  ATTRIBUTES (ADDRESS   '192.0.2.1')
  DEFAULT ROLE managerRole
   ENABLE
```

If user *user1* requests a trusted connection from IP address 192.0.2.1, the DB2 database system returns a warning (SQLSTATE 01679, SQLCODE +20360) to indicate that a trusted connection could not be established, and that user *user1* simply got a non-trusted connection. However, if user *user2* requests a trusted connection from IP address 192.0.2.1, the request is honored because the connection attributes are satisfied by the trusted context CTX1. Now that use *user2* has established a trusted connection, he or she can now acquire all the privileges and authorities associated with the trusted context role managerRole. These privileges and authorities may not be available to user *user2* outside the scope of this trusted connection

Using trusted contexts and trusted connections

You can establish an explicit trusted connection by making a request within an application when a connection to a DB2 database is established. The security administrator must have previously defined a trusted context, using the CREATE TRUSTED CONTEXT statement, with attributes matching those of the connection you are establishing (see Step 1, later).

Before you begin

The API you use to request an explicit trusted connection when you establish a connection depends on the type of application you are using (see the table in Step 2).

After you have established an explicit trusted connection, the application can switch the user ID of the connection to a different user ID using the appropriate API for the type of application (see the table in Step 3).

Procedure

1. The security administrator defines a trusted context in the server by using the CREATE TRUSTED CONTEXT statement. For example:

   ```
   CREATE TRUSTED CONTEXT MYTCX
         BASED UPON CONNECTION USING SYSTEM AUTHID NEWTON
         ATTRIBUTES (ADDRESS   '192.0.2.1')
         WITH USE FOR PUBLIC WITHOUT AUTHENTICATION
         ENABLE
   ```

2. To establish a trusted connection, use one of the following APIs in your application:

Option	Description
Application	**API**
CLI/ODBC	SQLConnect, SQLSetConnectAttr
XA CLI/ODBC	Xa_open
JAVA	getDB2TrustedPooledConnection, getDB2TrustedXAConnection

3. To switch to a different user, with or without authentication, use one of the following APIs in your application:

Option	Description
Application	**API**
CLI/ODBC	SQLSetConnectAttr
XA CLI/ODBC	SQLSetConnectAttr
JAVA	getDB2Connection, reuseDB2Connection
.NET	DB2Connection.ConnectionString keywords: TrustedContextSystemUserID and TrustedContextSystemPassword

   The switching can be done either with or without authenticating the new user ID, depending on the definition of the trusted context object associated with the explicit trusted connection. For example, suppose that the security administrator creates the following trusted context object:

   ```
   CREATE TRUSTED CONTEXT CTX1
    BASED UPON CONNECTION USING SYSTEM AUTHID USER1
    ATTRIBUTES (ADDRESS   '192.0.2.1')
     WITH USE FOR USER2 WITH AUTHENTICATION,
                USER3 WITHOUT AUTHENTICATION
     ENABLE
   ```

   Further, suppose that an explicit trusted connection is established. A request to switch the user ID on the trusted connection to USER3 without providing authentication information is allowed because USER3 is defined as a user of trusted context CTX1 for whom authentication is not required. However, a request to switch the user ID on the trusted connection to USER2 without providing authentication information will fail because USER2 is defined as a user of trusted context CTX1 for whom authentication information must be provided.

Example of establishing an explicit trusted connection and switching the user

In the following example, a middle-tier server needs to issue some database requests on behalf of an end-user, but does not have access to the end-user's credentials to establish a database connection on behalf of that end-user.

You can create a trusted context object on the database server that allows the middle-tier server to establish an explicit trusted connection to the database. After establishing an explicit trusted connection, the middle-tier server can switch the current user ID of the connection to a new user ID without the need to authenticate the new user ID at the database server. The following CLI code snippet demonstrates how to establish a trusted connection using the trusted context, MYTCX, defined in Step 1, earlier, and how to switch the user on the trusted connection without authentication.

```
int main(int argc, char *argv[])
{
 SQLHANDLE henv;         /* environment handle */
 SQLHANDLE hdbc1;       /* connection handle */
 char origUserid[10] = "newton";
  char password[10] = "test";
   char switchUserid[10] = "zurbie";
 char dbName[10] = "testdb";

// Allocate the handles
SQLAllocHandle( SQL_HANDLE_ENV, &henv );
SQLAllocHandle( SQL_HANDLE_DBC, &hdbc1 );

// Set the trusted connection attribute
SQLSetConnectAttr( hdbc1, SQL_ATTR_USE_TRUSTED_CONTEXT,
SQL_TRUE, SQL_IS_INTEGER );

// Establish a trusted connection
SQLConnect( hdbc1, dbName, SQL_NTS, origUserid, SQL_NTS,
password, SQL_NTS );

//Perform some work under user ID "newton"
. . . . . . . . . . .

 // Commit the work
SQLEndTran(SQL_HANDLE_DBC, hdbc1, SQL_COMMIT);


// Switch the user ID on the trusted connection
SQLSetConnectAttr( hdbc1,
SQL_ATTR_TRUSTED_CONTEXT_USERID, switchUserid,
SQL_IS_POINTER
);

//Perform new work using user ID "zurbie"
. . . . . . . . .

//Commit the work
SQLEndTranSQL_HANDLE_DBC, hdbc1, SQL_COMMIT);

// Disconnect from database
SQLDisconnect( hdbc1 );

 return 0;

}  /* end of main */
```

What to do next

When does the user ID actually get switched?

After the command to switch the user on the trusted connection is issued, the switch user request is not performed until the next statement is sent to the server. This is demonstrated by the following example where the **`list applications`** command shows the original user ID until the next statement is issued.

1. Establish an explicit trusted connection with USERID1.
2. Issue the switch user command, such as **`getDB2Connection`** for USERID2.
3. Run `db2 list applications`. It still shows that USERID1 is connected.
4. Issue a statement on the trusted connection, such as `executeQuery("values current sqlid")`, to perform the switch user request at the server.
5. Run `db2 list applications` again. It now shows that USERID2 is connected.

Chapter 60. Row and column access control (RCAC)

DB2 V10.1 introduces row and column access control (RCAC), as an additional layer of data security. Row and column access control is sometimes referred to as fine-grained access control or FGAC. RCAC controls access to a table at the row level, column level, or both. RCAC can be used to complement the table privileges model.

To comply with various government regulations, you might implement procedures and methods to ensure that information is adequately protected. Individuals in your organization are permitted access to only the subset of data that is required to perform their job tasks. For example, government regulations in your area might state that a doctor is authorized to view the medical records of their own patients, but not of other patients. The same regulations might also state that, unless a patient gives their consent, a healthcare provider is not permitted access to patient personal information, such as the patients home phone number.

You can use row and column access control to ensure that your users have access to only the data that is required for their work. For example, a hospital system running DB2 for Linux, UNIX, and Windows and RCAC can filter patient information and data to include only that data which a particular doctor requires. Other patients do not exist as far as the doctor is concerned. Similarly, when a patient service representative queries the patient table at the same hospital, they are able to view the patient name and telephone number columns, but the medical history column is masked for them. If data is masked, a NULL, or an alternate value is displayed, instead of the actual medical history.

Row and column access control, or RCAC, has the following advantages:

- No database user is inherently exempted from the row and column access control rules.

  Even higher level authorities such as users with DATAACCESS authority are not exempt from these rules. Only users with security administrator (SECADM) authority can manage row and column access controls within a database. Therefore, you can use RCAC to prevent users with DATAACCESS authority from freely accessing all data in a database.
- Table data is protected regardless of how a table is accessed via SQL.

  Applications, improvised query tools, and report generation tools are all subject to RCAC rules. The enforcement is data-centric.
- No application changes are required to take advantage of this additional layer of data security.

  That is, row and column level access controls are established and defined in a way that is not apparent to existing applications. However, RCAC represents an important shift in paradigm in the sense that it is no longer what is being asked but rather who is asking what. Result sets for the same query change based on the context in which the query was asked and there is no warning or error returned. This behavior is the exact intent of the solution. It means that application designers and DBAs must be conscious that queries do not see the whole picture in terms of the data in the table, unless granted specific permissions to do so.

Row and column access control (RCAC) rules

Row and column access control (RCAC) places access control at the table level around the data itself. SQL rules created on rows and columns are the basis of the implementation of this capability.

Row and column access control is an access control model in which a security administrator manages privacy and security policies. RCAC permits all users to access the same table, as opposed to alternative views of a table. RCAC does however, restrict access to the table based upon individual user permissions or rules as specified by a policy associated with the table. There are two sets of rules, one set operates on rows, and the other on columns.

- Row permission
  - A row permission is a database object that expresses a row access control rule for a specific table.
  - A row access control rule is an SQL search condition that describes what set of rows a user has access to.
- Column mask
  - A column mask is a database object that expresses a column access control rule for a specific column in a table.
  - A column access control rule is an SQL CASE expression that describes what column values a user is permitted to see and under what conditions.

Scenario: ExampleHMO using row and column access control

This scenario presents ExampleHMO, a national organization with a large and active list of patients, as a user of row and column access control. ExampleHMO uses row and column access control to ensure that their database policies reflect government regulatory requirements for privacy and security, as well as management business objectives.

Organizations that handle patient health information and their personal information, like ExampleHMO, must comply with government privacy and data protection regulations, for example the Health Insurance Portability and Accountability Act (HIPAA). These privacy and data protection regulations ensure that any sensitive patient medical or personal information is shared, viewed, and modified only by authorities who are privileged to do so. Any violation of the act results in huge penalties including civil and criminal suits.

ExampleHMO must ensure that the data stored in their database systems is secure and only privileged users have access to the data. According to typical privacy regulations, certain patient information can be accessed and modified by only privileged users.

Security policies

ExampleHMO implements a security strategy where data access to DB2 databases are made available according to certain security policies.

The security policies conform to government privacy and data protection regulations. The first column outlines the policies and the challenges faced by the organization, the second column outlines the DB2 row and column access control feature which addresses the challenge.

Security challenge	Row and column access control feature which addresses the security challenge
Limiting column access to only privileged users. For example, Jane, who is a drug researcher at a partner company, is not permitted to view sensitive patient medical information or personal data like their insurance number.	Column masks can be used to filter or hide sensitive data from Jane.
Limiting row access to only privileged users. Dr. Lee is only permitted to view patient information for his own patients, not all patients in the ExampleHMO system.	Row permissions can be implemented to control which user can view any particular row.
Restricting data on a need-to-know basis.	Row permissions can help with this challenge as well by restricting table level data at the user level.
Restricting other database objects like UDFs, triggers, views on RCAC secured data.	Row and column access control protects data at the data level. It is this data-centric nature of the row and column access control solution that enforces security policies on even database objects like UDFs, triggers, and views.

Database users and roles

In this scenario, a number of different people create, secure, and use ExampleHMO data. These people have different user rights and database authorities.

ExampleHMO implemented their security strategy to classify the way data is accessed from the database. Internal and external access to data is based on the separation of duties to users who access the data and their data access privileges. ExampleHMO created the following database roles to separate these duties:

PCP
For primary care physicians.

DRUG_RESEARCH
For researchers.

ACCOUNTING
For accountants.

MEMBERSHIP
For members who add patients for opt-in and opt-out.

PATIENT
For patients.

The following people create, secure, and use ExampleHMO data:

Alex
ExampleHMO Chief Security Administrator. He holds the SECADM authority.

Peter
ExampleHMO Database Administrator. He holds the DBADM authority.

Paul
ExampleHMO Database Developer. He has the privileges to create triggers and user-defined functions.

Dr. Lee
ExampleHMO Physician. He belongs to the PCP role.

Jane
Drug researcher at Innovative Pharmaceutical Company, a ExampleHMO partner. She belongs to the DRUG_RESEARCH role.

John
ExampleHMO Accounting Department. He belongs to the ACCOUNTING role.

Tom
ExampleHMO Membership Officer. He belongs to the MEMBERSHIP role.

Bob
ExampleHMO Patient. He belongs to the PATIENT role.

If you want to try any of the example SQL statements and commands presented in this scenario, create these user IDs with their listed authorities.

The following example SQL statements assume that the users have been created on the system. The SQL statements create each role and grant **SELECT** and **INSERT** permissions to the various tables in the ExampleHMO database to the users:

```
--Creating roles and granting authority

CREATE ROLE PCP;

CREATE ROLE DRUG_RESEARCH;

CREATE ROLE ACCOUNTING;

CREATE ROLE MEMBERSHIP;

CREATE ROLE PATIENT;

GRANT ROLE PCP TO USER LEE;
GRANT ROLE DRUG_RESEARCH TO USER JANE;
GRANT ROLE ACCOUNTING  TO USER JOHN;
GRANT ROLE MEMBERSHIP TO USER TOM;
GRANT ROLE PATIENT TO USER BOB;
```

Database tables

This scenario focuses on two tables in the ExampleHMO database: the PATIENT table and the PATIENTCHOICE table.

The PATIENT table stores basic patient information and health information. This scenario considers the following columns within the PATIENT table:

SSN
The patient's insurance number. A patient's insurance number is considered personal information.

NAME
The patient's name. A patient's name is considered personal information.

ADDRESS
The patient's address. A patient's address is considered personal information.

USERID
The patient's database ID.

PHARMACY
The patient's medical information.

ACCT_BALANCE
The patient's billing information.

PCP_ID
The patient's primary care physician database ID

The PATIENTCHOICE table stores individual patient opt-in and opt-out information which decides whether a patient wants to expose his health information to outsiders for research purposes in this table. This scenario considers the following columns within the PATIENTCHOICE table:

SSN
The patient's insurance number is used to match patients with their choices.

CHOICE
The name of a choice a patient can make.

VALUE
The decision made by the patients about the choice.

For example, the row 123-45-6789, drug_research, opt-in says that patient with SSN 123-45-6789 agrees to disclose their information for medical research purposes.

The following example SQL statements create the PATIENT, PATIENTCHOICE, and ACCT_HISTORY tables. Authority is granted on the tables and data is inserted:

```
--Patient table storing information regarding patient
CREATE TABLE PATIENT (
 SSN CHAR(11),
 USERID VARCHAR(18),
 NAME VARCHAR(128),
 ADDRESS VARCHAR(128),
 PHARMACY VARCHAR(250),
 ACCT_BALANCE DECIMAL(12,2) WITH DEFAULT,
 PCP_ID VARCHAR(18)
 );


--Patientchoice table which stores what patient opts
--to expose regarding his health information

CREATE TABLE PATIENTCHOICE (
 SSN CHAR(11),
 CHOICE VARCHAR(128),
 VALUE VARCHAR(128)
);

--Log table to track account balance
CREATE TABLE ACCT_HISTORY(
 SSN   CHAR(11),
 BEFORE_BALANCE DECIMAL(12,2),
 AFTER_BALANCE DECIMAL(12,2),
 WHEN  DATE,
 BY_WHO VARCHAR(20)
);


--Grant authority

GRANT SELECT, UPDATE ON TABLE PATIENT TO ROLE PCP;

GRANT SELECT ON TABLE PATIENT TO ROLE DRUG_RESEARCH;

GRANT SELECT, UPDATE ON TABLE PATIENT TO ROLE ACCOUNTING;
GRANT SELECT ON TABLE ACCT_HISTORY TO ROLE ACCOUNTING;

GRANT SELECT, UPDATE, INSERT ON TABLE PATIENT TO ROLE MEMBERSHIP;
GRANT INSERT ON TABLE PATIENTCHOICE TO ROLE MEMBERSHIP;

GRANT SELECT ON TABLE PATIENT TO ROLE PATIENT;

GRANT SELECT, ALTER ON TABLE PATIENT TO USER ALEX;

GRANT ALTER, SELECT ON TABLE PATIENT TO USER PAUL;
GRANT INSERT ON TABLE ACCT_HISTORY TO USER PAUL;

--Insert patient data

INSERT INTO PATIENT
  VALUES('123-55-1234', 'MAX', 'Max', 'First Strt', 'hypertension', 89.70,'LEE');
INSERT INTO PATIENTCHOICE
  VALUES('123-55-1234', 'drug-research', 'opt-out');


INSERT INTO PATIENT
  VALUES('123-58-9812', 'MIKE', 'Mike', 'Long Strt', null, 8.30,'JAMES');
INSERT INTO PATIENTCHOICE
  VALUES('123-58-9812', 'drug-research', 'opt-out');


INSERT INTO PATIENT
  VALUES('123-11-9856', 'SAM', 'Sam', 'Big Strt', null, 0.00,'LEE');
INSERT INTO PATIENTCHOICE
  VALUES('123-11-9856', 'drug-research', 'opt-in');


INSERT INTO PATIENT
  VALUES('123-19-1454', 'DUG', 'Dug', 'Good Strt', null, 0.00,'JAMES');
INSERT INTO PATIENTCHOICE
  VALUES('123-19-1454', 'drug-research', 'opt-in');
```

Security administration

Security administration and the security administrator (SECADM) role play important parts in securing patient and company data at ExampleHMO. At ExampleHMO, management decided that different people hold database administration authority and security administration authority.

The management team at ExampleHMO decides to create a role for administering access to their data. The team also decides that even users with DATAACCESS authority are not able to view protected health and personal data by default.

The management team selects Alex to be the sole security administrator for ExampleHMO. From now on, Alex controls all data access authority. With this authority, Alex defines security rules such as row permissions, column masks, and whether functions and triggers are secure or not. These rules control which users have access to any given data under his control.

After Peter, the database administrator, creates the required tables and sets up the required roles, duties are separated. The database administration and security administration duties are separated by making Alex the security administrator.

Peter connects to the database and grants Alex SECADM authority. Peter can grant SECADM authority since he currently holds the DBADM, DATAACCESS, and SECADM authorities.

```
-- To seperate duties of security administrator from system administrator,
-- the SECADMN Peter grants SECADM authority to user Alex.

GRANT SECADM ON DATABASE TO USER ALEX;
```

Alex, after receiving the SECADM authority, connects to the database and revokes the security administrator privilege from Peter. The duties are now separated and Alex becomes the sole authority to grant data access to others within and outside ExampleHMO. The following SQL statement shows how Alex revoked SECADM authority from Peter:

```
--revokes the SECADMIN authority for Peter

REVOKE SECADM ON DATABASE FROM USER PETER;
```

Row permissions

Alex, the security administrator, starts to restrict data access on the ExampleHMO database by using row permissions, a part of row and column access control. Row permissions filter the data returned to users by row.

Patients are permitted to view their own data. A physician is permitted to view the data of all his patients, but not the data of patients who see other physicians. Users belonging to the MEMBERSHIP, ACCOUNTING, or DRUG_RESEARCH roles can access all patient information. Alex, the security administrator, is asked to implement these permissions to restrict who can see any given row on a need-to-know basis.

Row permissions restrict or filter rows based on the user who has logged on to the database. At ExampleHMO, the row permissions create a horizontal data restriction on the table named PATIENT.

Alex implements the following row permissions so that a user in each role is restricted to view a result set that they are privileged to view:

```
CREATE PERMISSION ROW_ACCESS ON PATIENT
-------------------------------------------------------
-- Accounting information:
-- ROLE PATIENT is allowed to access his or her own row
-- ROLE PCP is allowed to access his or her patients' rows
-- ROLE MEMBERSHIP, ACCOUNTING, and DRUG_RESEARCH are
-- allowed to access all rows
------------------------------------------------------------
FOR ROWS WHERE(VERIFY_ROLE_FOR_USER(SESSION_USER,'PATIENT') = 1
AND
PATIENT.USERID = SESSION_USER) OR
(VERIFY_ROLE_FOR_USER(SESSION_USER,'PCP') = 1
AND
PATIENT.PCP_ID = SESSION_USER) OR
 (VERIFY_ROLE_FOR_USER(SESSION_USER,'MEMBERSHIP') = 1 OR
 VERIFY_ROLE_FOR_USER(SESSION_USER,'ACCOUNTING') = 1 OR
 VERIFY_ROLE_FOR_USER(SESSION_USER, 'DRUG_RESEARCH') = 1)
ENFORCED FOR ALL ACCESS
ENABLE;
```

Alex observes that even after creating a row permission, all data can still be viewed by the other employees. A row permission is not applied until it is activated on the table for which it was defined. Alex must now activate the permission:

```
--Activate row access control to implement row permissions

ALTER TABLE PATIENT ACTIVATE ROW ACCESS CONTROL;
```

Column masks

Alex, the security administrator, further restricts data access on the ExampleHMO database by using column masks, a part of row and column access control. Column masks hide data returned to users by column unless they are permitted to view the data.

Patient payment details must only be accessible to the users in the accounts department. The account balance must not be seen by any other database users. Alex is asked to prevent access by anyone other than users belonging to the ACCOUNTING role.

Alex implements the following column mask so that a user in each role is restricted to view a result set that they are privileged to view:

```
--Create a Column MASK ON ACCT_BALANCE column on the PATIENT table

CREATE MASK ACCT_BALANCE_MASK ON PATIENT FOR
------------------------------------------------------------
-- Accounting information:
-- Role ACCOUNTING is allowed to access the full information
-- on column ACCT_BALANCE.
-- Other roles accessing this column will strictly view a
-- zero value.
------------------------------------------------------------
COLUMN ACCT_BALANCE RETURN
 CASE WHEN VERIFY_ROLE_FOR_USER(SESSION_USER,'ACCOUNTING') = 1
   THEN ACCT_BALANCE
  ELSE 0.00
 END
ENABLE;
```

Alex observes that even after creating a column mask, the data can still be viewed by the other employees. A column mask is not applied until it is activated on the table for which it was defined. Alex must now activate the mask:

```
--Activate column access control to implement column masks

ALTER TABLE PATIENT ACTIVATE COLUMN ACCESS CONTROL;
```

Alex is asked by management to hide the insurance number of the patients. Only a patient, physician, accountant, or people in the MEMBERSHIP role can view the SSN column.

Also, to protect the PHARMACY detail of a patient, the information in the PHARMACY column must only be viewed by a drug researcher or a physician. Drug researchers can see the data only if the patient has agreed to disclose the information.

Alex implements the following column masks so that a user in each role is restricted to view a result set that they are privileged to view:

```
CREATE MASK SSN_MASK ON PATIENT FOR
----------------------------------------------------
-- Personal contact information:
-- Roles PATIENT, PCP, MEMBERSHIP, and ACCOUNTING are allowed
-- to access the full information on columns SSN, USERID, NAME,
-- and ADDRESS. Other roles accessing these columns will
-- strictly view a masked value.
-----------------------------------------------------
COLUMN SSN RETURN
 CASE WHEN
  VERIFY_ROLE_FOR_USER(SESSION_USER,'PATIENT') = 1 OR
  VERIFY_ROLE_FOR_USER(SESSION_USER,'PCP') = 1 OR
  VERIFY_ROLE_FOR_USER(SESSION_USER,'MEMBERSHIP') = 1 OR
  VERIFY_ROLE_FOR_USER(SESSION_USER,'ACCOUNTING') = 1
 THEN SSN
  ELSE CHAR('XXX-XX-' || SUBSTR(SSN,8,4)) END
ENABLE;

CREATE MASK PHARMACY_MASK ON PATIENT FOR
--------------------------------------------------------
-- Medical information:
-- Role PCP is allowed to access the full information on
-- column PHARMACY.
-- For the purposes of drug research, Role DRUG_RESEARCH can
-- conditionally see a patient's medical information
-- provided that the patient has opted-in.
-- In all other cases, null values are rendered as column
-- values.
----------------------------------------------------
COLUMN PHARMACY RETURN
  CASE WHEN
  VERIFY_ROLE_FOR_USER(SESSION_USER,'PCP') = 1 OR
  (VERIFY_ROLE_FOR_USER(SESSION_USER,'DRUG_RESEARCH')=1
  AND
   EXISTS (SELECT 1 FROM PATIENTCHOICE C
 WHERE PATIENT.SSN = C.SSN AND C.CHOICE = 'drug-research' AND C.VALUE = 'opt-in'))
  THEN PHARMACY
  ELSE NULL
END
ENABLE;
```

Alex observes that after creating these two column masks that the data is only viewable to the intended users. The PATIENT table already had column access control activated.

Inserting data

When a new patient is admitted for treatment in the hospital, the new patient record must be added to the ExampleHMO database.

Bob is a new patient, and his records must be added to the ExampleHMO database. A user with the required security authority must create the new record for Bob. Tom, from the ExampleHMO membership department, with the MEMBERSHIP role, enrolls Bob as a new member. After connecting to the ExampleHMO database, Tom runs the following SQL statements to add Bob to the ExampleHMO database:

```
INSERT INTO PATIENT
  VALUES('123-45-6789', 'BOB', 'Bob', '123 Some St.', 'hypertension', 9.00,'LEE');
INSERT INTO PATIENTCHOICE
  VALUES('123-45-6789', 'drug-research', 'opt-in');
```

Tom confirmed that Bob was added to the database by querying the same from the PATIENT table in the ExampleHMO database:

```
Select * FROM PATIENT WHERE NAME = 'Bob';

SSN         USERID    NAME    ADDRESS       PHARMACY     ACCT_BALANCE  PCP_ID

----------- --------- ------- ------------- ------------ ------------- ------
123-45-6789 BOB       Bob     123 Some St.  XXXXXXXXXXX  0.00          LEE
```

Updating data

While in the hospital, Bob gets his treatment changed. As a result his records in the ExampleHMO database need updating.

Dr. Lee, who is Bob's physician, advises a treatment change and changes Bob's medicine. Bob's record in the ExampleHMO systems must be updated. The row permission rules set in the ExampleHMO database specify that anyone who cannot view the data in a row cannot update the data in that row. Since Bob's PCPID contains Dr. Lee's ID, and the row permission is set, Dr. Lee can both view, and update Bob's record using the following example SQL statement:

```
UPDATE PATIENT SET PHARMACY = 'codeine' WHERE NAME = 'Bob';
```

Dr. Lee checks the update:

```
Select * FROM PATIENT WHERE NAME = 'Bob';

SSN         USERID    NAME    ADDRESS      PHARMACY    ACCT_BALANCE   PCP_ID

----------- --------- ------- ------------ ----------- -------------- ------
123-45-6789 BOB       Bob     123 Some St. codeine     0.00           LEE
```

Dug is a patient who is under the care of Dr. James, one of Dr. Lee's colleagues. Dr. Lee attempts the same update on the record for Dug:

```
UPDATE PATIENT SET PHARMACY = 'codeine' WHERE NAME = 'Dug';
SQL0100W  No row was found for FETCH, UPDATE or DELETE; or the result of a query
 is an empty table. SQLSTATE=02000
```

Since Dug's PCPID does not contain Dr. Lee's ID, and the row permission is set, Dr. Lee cannot view, or update Dug's record.

Reading data

With row and column access control, people in different roles can have different result sets from the same database queries. For example, Peter, the database administrator with DATAACCESS authority, cannot see any data on the PATIENT table.

Peter, Bob, Dr. Lee, Tom, Jane, and John each connect to the database and try the following SQL query:

```
SELECT SSN, USERID, NAME, ADDRESS, PHARMACY, ACCT_BALANCE, PCP_ID FROM PATIENT;
```

Results of the query vary according to who runs the query. The row and column access control rules created by Alex are applied on these queries.

Here is the result set Peter sees:

```
SSN         USERID    NAME      ADDRESS     PHARMACY    ACC_BALANCE PCP_ID

----------- --------- --------- ----------- ----------- ----------- -----------

  0 record(s) selected.
```

Even though there is data in the table and Peter is the database administrator, he lacks the authority to see all data.

Here is the result set Bob sees:

```
SSN         USERID    NAME     ADDRESS     PHARMACY    ACC_BALANCE PCP_ID

----------- --------- --------- ----------- ----------- ----------- ------
123-45-6789 BOB       Bob      123 Some St.XXXXXXXXXXX 0.00        LEE

1 record(s) selected.
```

Bob, being a patient, can only see his own data. Bob belongs to the PATIENT role. The PHARMACY and ACC_BALANCE column data have been hidden from him.

Here is the result set Dr. Lee sees:

```
SSN         USERID   NAME     ADDRESS     PHARMACY            ACC_BALANCE PCP_ID

----------- -------- -------- ----------- ------------------- ----------- ------
123-55-1234 MAX      Max      First Strt  hypertension        0.00        LEE
123-11-9856 SAM      Sam      Big Strt    High blood pressure 0.00        LEE
123-45-6789 BOB      Bob      123 Some St.codeine             0.00        LEE

3 record(s) selected.
```

Dr. Lee can see only the data for patients under his care. Dr. Lee belongs to the PCP role. The ACC_BALANCE column data is hidden from him.

Here is the result set Tom sees:

```
SSN         USERID   NAME     ADDRESS     PHARMACY    ACC_BALANCE PCP_ID

----------- -------- -------- ----------- ----------- ----------- -----------
123-55-1234 MAX      Max      First Strt  XXXXXXXXXXX 0.00        LEE
123-58-9812 MIKE     Mike     Long Strt   XXXXXXXXXXX 0.00        JAMES
123-11-9856 SAM      Sam      Big Strt    XXXXXXXXXXX 0.00        LEE
123-19-1454 DUG      Dug      Good Strt   XXXXXXXXXXX 0.00        JAMES
123-45-6789 BOB      Bob      123 Some St.XXXXXXXXXXX 0.00        LEE

5 record(s) selected.
```

Tom can see all members. Tom belongs to the membership role. He is not privileged to see any data in the PHARMACY and ACC_BALANCE columns.

Here is the result set Jane sees:

```
SSN         USERID   NAME     ADDRESS     PHARMACY            ACC_BALANCE PCP_ID

----------- -------- -------- ----------- ------------------- ----------- -------
XXX-XX-1234 MAX      Max      First Strt  XXXXXXXXXXX         0.00        LEE
XXX-XX-9812 MIKE     Mike     Long Strt   XXXXXXXXXXX         0.00        JAMES
XXX-XX-9856 SAM      Sam      Big Strt    High blood pressure 0.00        LEE
XXX-XX-1454 DUG      Dug      Good Strt   Influenza           0.00        JAMES
XXX-XX-6789 BOB      Bob      123 Some St.codeine             0.00        LEE

5 record(s) selected.
```

Jane can see all members. She belongs to the DRUG_RESEARCH role. The SSN and ACC_BALANCE column data are hidden from her. The PHARMACY data is only available if the patients have opted-in to share their data with drug research companies.

Here is the result set John sees:

```
SSN         USERID   NAME     ADDRESS     PHARMACY    ACC_BALANCE PCP_ID

----------- -------- -------- ----------- ----------- ----------- --------
123-55-1234 MAX      Max      First Strt  XXXXXXXXXXX 89.70       LEE
123-58-9812 MIKE     Mike     Long Strt   XXXXXXXXXXX 8.30        JAMES
123-11-9856 SAM      Sam      Big Strt    XXXXXXXXXXX 0.00        LEE
123-19-1454 DUG      Dug      Good Strt   XXXXXXXXXXX 0.00        JAMES
123-45-6789 BOB      Bob      123 Some St.XXXXXXXXXXX 9.00        LEE

5 record(s) selected.
```

John can see all members. He belongs to the ACCOUNTING role. The PHARMACY column data is hidden from him.

Creating views

Views can be created on tables that have row and column access control defined. Alex, the security administrator, is asked to create a view on the PATIENT table that medical researchers can use.

Researchers, that have a partnership with ExampleHMO, can have access to limited patient data if patients have opted-in to permit this access. Alex and the IT team are asked to create a view to list only specific information related to research of the patient. The report must contain the patient insurance number, name of the patient and the disclosure option chosen by the patient.

The view created fetches the patient basic information and the health condition disclosure option. This view ensures that patient information is protected and fetched only with their permission for any other purpose.

Alex and the IT team implement the following view:

```
CREATE VIEW PATIENT_INFO_VIEW AS
SELECT P.SSN, P.NAME FROM PATIENT P, PATIENTCHOICE C
WHERE P.SSN = C.SSN AND
   C.CHOICE = 'drug-research' AND
   C.VALUE = 'opt-in';
```

After Alex and his team create the view, users can query the view. They see data according to the row and column access control rules defined on the base tables on which the view is created.

Alex sees the following result-set from the following query on the view:

```
SELECT SSN, NAME FROM PATIENT_INFO_VIEW;

SSN         NAME
----------- ----------

  0 record(s) selected.
```

Dr. Lee sees the following result-set from the following query on the view:

```
SELECT SSN, NAME FROM PATIENT_INFO_VIEW;

SSN         NAME
----------- ----------
123-11-9856 Sam
123-45-6789 Bob

  2 record(s) selected.
```

Bob sees the following result-set from the following query on the view:

```
SELECT SSN, NAME FROM PATIENT_INFO_VIEW;

SSN         NAME
----------- ----------
123-45-6789 Bob

  1 record(s) selected.
```

Secure functions

Functions must be deemed secure before they can be called within row and column access control definitions. Alex, the security administrator, discusses how Paul, a database developer at ExampleHMO, can create a secure function for his new accounting application.

After the privacy and security policy went into effect at ExampleHMO, Alex is notified that the accounting department has developed a powerful accounting application. ExampleHMOAccountingUDF is a SQL scalar user-defined function (UDF) that is used in the column mask ACCT_BALANCE_MASK on the PATIENT.ACCT_BALANCE table and row.

Only UDFs that are secure can be invoked within a column mask. Alex first discusses the UDF with Paul, who wrote the UDF, to ensure the operation inside the UDF is secure.

When Alex is satisfied that the function is secure, he grants a system privilege to Paul so Paul can alter the UDF to be secure:

```
GRANT CREATE_SECURE_OBJECT ON DATABASE TO USER PAUL;
```

To create a secured UDF, or alter a UDF to be secured, a developer must be granted CREATE_SECURE_OBJECT authority.

Paul creates the function:

```
CREATE FUNCTION EXAMPLEHMOACCOUNTINGUDF(X DECIMAL(12,2))
       RETURNS DECIMAL(12,2)
       LANGUAGE SQL
       CONTAINS SQL
       DETERMINISTIC
       NO EXTERNAL ACTION
       RETURN X*(1.0 + RAND(X));
```

Paul alters the function so it is secured:

```
ALTER FUNCTION EXAMPLEHMOACCOUNTINGUDF SECURED;
```

Alex now drops and recreates the mask ACC_BALANCE_MASK so the new UDF is used:

```
--Drop the mask to recreate

DROP MASK ACCT_BALANCE_MASK;



CREATE MASK EXAMPLEHMO.ACCT_BALANCE_MASK ONPATIENT FOR
------------------------------------------------------------
-- Accounting information:
-- Role ACCOUNTING is allowed to invoke the secured UDF
-- ExampleHMOAccountingUDFL passing column ACCT_BALANCE as
-- the input argument
-- Other ROLEs accessing this column will strictly view a
-- zero value.
------------------------------------------------------------
COLUMN ACCT_BALANCE RETURN
CASE WHEN VERIFY_ROLE_FOR_USER(SESSION_USER,'ACCOUNTING') = 1
THEN EXAMPLEHMOACCOUNTINGUDF(ACCT_BALANCE)
ELSE 0.00
END
ENABLE;
```

Dr. Lee, who has the PCP role, must call a drug analysis user-defined function. DrugUDF returns patient drug information. In the past, Dr. Lee issues a SELECT statement that calls DrugUDF and receives the result set quickly. After the PATIENT table has been protected with row and column access control, the same query takes more time to return a result set.

Dr. Lee consults with the ExampleHMO IT staff and Alex, the security administrator, about this performance degradation. Alex tells Dr. Lee, if the UDF is not secure, the query cannot be optimized as well and it takes longer to return a result set.

Alex looks into the UDF with Dr. Lee and the owner, Paul, to ensure the operation inside the UDF is secure. Alex asks Paul to alter the UDF to be secure as Paul still has the CREATE_SECURE_OBJECT privilege granted by Alex:

```
--Function for ExampleHMO Pharmacy department

CREATE FUNCTION DRUGUDF(PHARMACY VARCHAR(5000))
      RETURNS VARCHAR(5000)
      NO EXTERNAL ACTION
      BEGIN ATOMIC
       IF PHARMACY IS NULL THEN
          RETURN NULL;
        ELSE
          RETURN 'Normal';
        END IF;
      END;

--Secure the UDF

ALTER FUNCTION DRUGUDF SECURED;

--Grant execute permissions to Dr.Lee

GRANT EXECUTE ON FUNCTION DRUGUDF TO USER LEE;
```

Dr. Lee can issue the query and the query can be optimized as expected:

```
--Querying after the function is secured

SELECT PHARMACY FROM PATIENT
      WHERE DRUGUDF(PHARMACY) = 'Normal' AND SSN = '123-45-6789';

PHARMACY
--------
codeine

  1 record(s) selected.
```

Secure triggers

Triggers defined on a table with row or column access control activated must be secure. Alex, the security administrator, discusses how Paul, a database developer at ExampleHMO, can create a secure trigger for his new accounting application.

Alex speaks to the accounting department and learns that an AFTER UPDATE trigger is needed for the PATIENT table. This trigger monitors the history of the ACCT_BALANCE column.

Alex explains to Paul, who has the necessary privileges to create the trigger, that any trigger defined on a row and column access protected table must be marked secure. Paul and Alex review the action of the new trigger and deem it to be secure.

ExampleHMO_ACCT_BALANCE_TRIGGER monitors the ACCT_BALANCE column in the PATIENT table. Every time that column is updated, the trigger is fired, and inserts the current account balance details into the ACCT_HISTORY table.

Paul creates the trigger:

```
CREATE TRIGGER HOSPITAL.NETHMO_ACCT_BALANCE_TRIGGER
  AFTER UPDATE OF ACCT_BALANCE ON PATIENT
  REFERENCING OLD AS O NEW AS N
  FOR EACH ROW MODE DB2SQL SECURED
  BEGIN  ATOMIC
  INSERT INTO ACCT_HISTORY
  (SSN, BEFORE_BALANCE, AFTER_BALANCE, WHEN, BY_WHO)
  VALUES(O.SSN, O.ACCT_BALANCE, N.ACCT_BALANCE,
  CURRENT TIMESTAMP, SESSION_USER);
END;
```

John, from the accounting department, must update the account balance for the patient Bob whose SSN is '123-45-6789'.

John looks at the data for Bob before running the update:

```
SELECT ACCT_BALANCE FROM PATIENT WHERE SSN = '123-45-6789';

ACCT_BALANCE
--------------
9.00

  1 record(s) selected.

SELECT * FROM ACCT_HISTORY WHERE SSN = '123-45-6789';

SSN         BEFORE_BALANCE AFTER_BALANCE  WHEN       BY_WHO
----------- -------------- -------------- ---------- --------------------

  0 record(s) selected.
```

John then runs the update:

```
UPDATE PATIENT SET ACCT_BALANCE = ACCT_BALANCE * 0.9 WHERE SSN = '123-45-6789';
```

Since there is a trigger defined on the PATIENT table, the update fires the trigger. Since the trigger is defined SECURED, the update completes successfully. John looks at the data for Bob after running the update:

```
SELECT ACCT_BALANCE FROM PATIENT WHERE SSN = '123-45-6789';

ACCT_BALANCE
--------------
8.10

  1 record(s) selected.

SELECT * FROM ACCT_HISTORY WHERE SSN = '123-45-6789';

SSN         BEFORE_BALANCE AFTER_BALANCE  WHEN       BY_WHO
----------- -------------- -------------- ---------- --------------------
123-45-6789 9.00           8.10           2010-10-10 JOHN
  1 record(s) selected.
```

Revoking authority

Alex, as security administrator, is responsible for controlling who can create secure objects. When developers are done creating secure objects, Alex revokes their authority on the database.

Paul, the database developer, is done with development activities. Alex immediately revokes the create authority from Paul:

```
REVOKE CREATE_SECURE_OBJECT ON DATABASE FROM USER PAUL;
```

If Paul must create secure objects in the future, he must speak to Alex to have the create authority granted again.

Chapter 61. Label-based access control (LBAC)

Label-based access control (LBAC) greatly increases the control you have over who can access your data. LBAC lets you decide exactly who has write access and who has read access to individual rows and individual columns.

What LBAC does

The LBAC capability is very configurable and can be tailored to match your particular security environment. All LBAC configuration is performed by a *security administrator*, which is a user that has been granted the SECADM authority.

A security administrator configures the LBAC system by creating security label components. A *security label component* is a database object that represents a criterion you want to use to determine if a user should access a piece of data. For example, the criterion can be whether the user is in a certain department, or whether they are working on a certain project. A *security policy* describes the criteria that will be used to decide who has access to what data. A security policy contains one or more security label components. Only one security policy can be used to protect any one table but different tables can be protected by different security policies.

After creating a security policy, a security administrator creates objects, called *security labels* that are part of that policy. Security labels contain security label components. Exactly what makes up a security label is determined by the security policy and can be configured to represent the criteria that your organization uses to decide who should have access to particular data items. If you decide, for example, that you want to look at a person's position in the company and what projects they are part of to decide what data they should see, then you can configure your security labels so that each label can include that information. LBAC is flexible enough to let you set up anything from very complicated criteria, to a very simple system where each label represents either a "high" or a "low" level of trust.

Once created, a security label can be associated with individual columns and rows in a table to protect the data held there. Data that is protected by a security label is called *protected data*. A security administrator allows users access to protected data by granting them security labels. When a user tries to access protected data, that user's security label is compared to the security label protecting the data. The protecting label will block some security labels and not block others.

A user, a role, or a group is allowed to hold security labels for multiple security policies at once. For any given security policy, however, a use, a role, or a group can hold at most one label for read access and one label for write access.

A security administrator can also grant exemptions to users. An *exemption* allows you to access protected data that your security labels might otherwise prevent you from accessing. Together your security labels and exemptions are called your *LBAC credentials*.

If you try to access a protected column that your LBAC credentials do not allow you to access then the access will fail and you will get an error message.

If you try to read protected rows that your LBAC credentials do not allow you to read then DB2 acts as if those rows do not exist. Those rows cannot be selected as part of any SQL statement that you run, including SELECT, UPDATE, or DELETE. Even the aggregate functions ignore rows that your LBAC credentials do not allow you to read. The COUNT(*) function, for example, will return a count only of the rows that you have read access to.

Views and LBAC

You can define a view on a protected table the same way you can define one on a non-protected table. When such a view is accessed the LBAC protection on the underlying table is enforced. The LBAC credentials used are those of the session authorization ID. Two users accessing the same view might see different rows depending on their LBAC credentials.

Referential integrity constraints and LBAC

The following rules explain how LBAC rules are enforced in the presence of referential integrity constraints:

- **Rule 1**: The LBAC read access rules are NOT applied for internally generated scans of child tables. This is to avoid having orphan children.
- **Rule 2**: The LBAC read access rules are NOT applied for internally generated scans of parent tables
- **Rule 3**: The LBAC write rules are applied when a CASCADE operation is performed on child tables. For example, If a user deletes a parent, but cannot delete any of the children because of an LBAC write rule violation, then the delete should be rolled-back and an error raised.

Storage overhead when using LBAC

When you use LBAC to protect a table at the row level, the additional storage cost is the cost of the row security label column. This cost depends on the type of security label chosen. For example, if you create a security policy with two components to protect a table, a security label from that security policy will occupy 16 bytes (8 bytes for each component). Because the row security label column is treated as a not nullable VARCHAR column, the total cost in this case would be 20 bytes per row. In general, the total cost per row is (*N**8 + 4) bytes where *N* is the number of components in the security policy protecting the table.

When you use LBAC to protect a table at the column level, the column security label is meta-data (that is, it is stored together with the column's meta-data in the SYSCOLUMNS catalog table). This meta-data is simply the ID of the security label protecting the column. The user table does not incur any storage overhead in this case.

What LBAC does not do

- LBAC will never allow access to data that is forbidden by discretionary access control.

  Example: If you do not have permission to read from a table then you will not be allowed to read data from that table--even the rows and columns to which LBAC would otherwise allow you access.
- Your LBAC credentials only limit your access to protected data. They have no effect on your access to unprotected data.

- LBAC credentials are not checked when you drop a table or a database, even if the table or database contains protected data.
- LBAC credentials are not checked when you back up your data. If you can run a backup on a table, which rows are backed up is not limited in any way by the LBAC protection on the data. Also, data on the backup media is not protected by LBAC. Only data in the database is protected.
- LBAC cannot be used to protect any of the following types of tables:
  - A staging table
  - A table that a staging table depends on
  - A typed table
- LBAC protection cannot be applied to a nickname.

LBAC tutorial

A tutorial leading you through the basics of using LBAC is available online at http://www.ibm.com/developerworks/data and is called DB2 Label-Based Access Control, a practical guide.

LBAC security policies

The security administrator uses a security policy to define criteria that determine who has write access and who has read access to individual rows and individual columns of tables.

A security policy includes this information:

- What security label components are used in the security labels that are part of the policy
- What rules are used when comparing those security label components
- Which of certain optional behaviors are used when accessing data protected by the policy
- What additional security labels and exemptions are to be considered when enforcing access to data protected by the security policy. For example, the option to consider or not to consider security labels granted to roles and groups is controlled through the security policy.

Every protected table must have one and only one security policy associated with it. Rows and columns in that table can only be protected with security labels that are part of that security policy and all access of protected data follows the rules of that policy. You can have multiple security policies in a single database but you cannot have more than one security policy protecting any given table.

Creating a security policy

You must be a security administrator to create a security policy. You create a security policy with the SQL statement CREATE SECURITY POLICY. The security label components listed in a security policy must be created before the CREATE SECURITY POLICY statement is executed. The order in which the components are listed when a security policy is created does not indicate any sort of precedence or other relationship among the components but it is important to know the order when creating security labels with built-in functions like SECLABEL.

From the security policy you have created, you can create security labels to protect your data.

Altering a security policy

A security administrator can use the ALTER SECURITY POLICY statement to modify a security policy.

Dropping a security policy

You must be a security administrator to drop a security policy. You drop a security policy using the SQL statement DROP.

You cannot drop a security policy if it is associated with (added to) any table.

LBAC security label components

A *security label component* is a database object that is part of label-based access control (LBAC). You use security label components to model your organization's security structure.

A security label component can represent any criteria that you might use to decide if a user should have access to a given piece of data. Typical examples of such criteria include:

- How well trusted the user is
- What department the user is in
- Whether the user is involved in a particular project

Example: If you want the department that a user is in to affect which data they can access, you could create a component named dept and define elements for that component that name the various departments in your company. You would then include the component dept in your security policy.

An *element* of a security label component is one particular "setting" that is allowed for that component.

Example: A security label component that represents a level of trust might have the four elements: Top Secret, Secret, Classified, and Unclassified.

Creating a security label component

You must be a security administrator to create a security label component. You create security label components with the SQL statement CREATE SECURITY LABEL COMPONENT.

When you create a security label component you must provide:

- A name for the component
- What type of component it is (ARRAY, TREE, or SET)
- A complete list of allowed elements
- For types ARRAY and TREE you must describe how each element fits into the structure of the component

After creating your security label components, you can create a security policy based on these components. From this security policy, you can create security labels to protect your data.

Types of components

There are three types of security label components:

- TREE: Each element represents a node in a tree structure
- ARRAY: Each element represents a point on a linear scale
- SET: Each element represents one member of a set

The types are used to model the different ways in which elements can relate to each other. For example, if you are creating a component to describe one or more departments in a company you would probably want to use a component type of TREE because most business structures are in the form of a tree. If you are creating a component to represent the level of trust that a person has, you would probably use a component of type ARRAY because for any two levels of trust, one will always be higher than the other.

The details of each type, including detailed descriptions of the relationships that the elements can have with each other, are described in their own section.

Altering security label components

The security administrator can use the ALTER SECURITY LABEL COMPONENT statement to modify a security label component.

Dropping a security label component

You must be a security administrator to drop a security label component. You drop a security label component with the SQL statement DROP.

LBAC security labels

In label-based access control (LBAC) a *security label* is a database object that describes a certain set of security criteria. Security labels are applied to data in order to protect the data. They are granted to users to allow them to access protected data.

When a user tries to access protected data, their security label is compared to the security label that is protecting the data. The protecting security label will block some security labels and not block others. If a user's security label is blocked then the user cannot access the data.

Every security label is part of exactly one security policy and includes one value for each component in that security policy. A *value* in the context of a security label component is a list of zero or more of the elements allowed by that component. Values for ARRAY type components can contain zero or one element, values for other types can have zero or more elements. A value that does not include any elements is called an *empty value.*

Example: If a TREE type component has the three elements Human Resources, Sales, and Shipping then these are some of the valid values for that component:

- Human Resources (or any of the elements by itself)
- Human Resources, Shipping (or any other combination of the elements as long as no element is included more than once)
- *An empty value*

Whether a particular security label will block another is determined by the values of each component in the labels and the LBAC rule set that is specified in the security policy of the table. The details of how the comparison is made are given in the topic that discusses how LBAC security labels are compared.

When security labels are converted to a text string they use the format described in the topic that discusses the format for security label values.

Creating security labels

You must be a security administrator to create a security label. You create a security label with the SQL statement CREATE SECURITY LABEL. When you create a security label you provide:

- A name for the label
- The security policy that the label is part of
- Values for one or more of the components included in the security policy

Any components for which a value is not specified is assumed to have an empty value. A security label must have at least one non-empty value.

Altering security labels

Security labels cannot be altered. The only way to change a security label is to drop it and re-create it. However, the *components* of a security label can be modified by a security administrator (using the ALTER SECURITY LABEL COMPONENT statement).

Dropping security labels

You must be a security administrator to drop a security label. You drop a security label with the SQL statement DROP. You cannot drop a security label that is being used to protect data anywhere in the database or that is currently held by one or more users.

Granting security labels

You must be a security administrator to grant a security label to a user, a group, or a role. You grant a security label with the SQL statement GRANT SECURITY LABEL. When you grant a security label you can grant it for read access, for write access, or for both read and write access. A user, a group, or a role cannot hold more than one security label from the same security policy for the same type of access.

Revoking security labels

You must be a security administrator to revoke a security label from a user, group, or role. To revoke a security label, use the SQL statement REVOKE SECURITY LABEL.

Data types compatible with security labels

Security labels have a data type of SYSPROC.DB2SECURITYLABEL. Data conversion is supported between SYSPROC.DB2SECURITYLABEL and VARCHAR(128) FOR BIT DATA.

Determining the security labels held by users

You can use the following query to determine the security labels that are held by users:

```
SELECT A.grantee, B.secpolicyname, c.seclabelname
FROM syscat.securitylabelaccess A, syscat.securitypolicies B, syscat.securitylabels C
WHERE A.seclabelid = C.seclabelid and B.secpolicyid = C.secpolicyid
```

Format for security label values

Sometimes the values in a security label are represented in the form of a character string, for example when using the built-in function SECLABEL.

When the values in a security label are represented as a string, they are in the following format:

- The values of the components are listed from left to right in the same order that the components are listed in the CREATE SECURITY POLICY statement for the security policy
- An element is represented by the name of that element
- Elements for different components are separated by a colon (:)
- If more than one element are given for the same component the elements are enclosed in parentheses (()) and are separated by a comma (,)
- Empty values are represented by a set of empty parentheses (())

Example: A security label is part of a security policy that has these three components in this order: Level, Department, and Projects. The security label has these values:

Table 137. Example values for a security label

Component	Values
Level	Secret
Department	*Empty value*
Projects	• Epsilon 37 • Megaphone • Cloverleaf

This security label values look like this as a string:

```
'Secret:():(Epsilon 37,Megaphone,Cloverleaf)'
```

How LBAC security labels are compared

When you try to access data protected by label-based access control (LBAC), your LBAC credentials are compared to one or more security labels to see if the access is blocked. Your LBAC credentials are any security labels you hold plus any exemptions that you hold.

There are only two types of comparison that can be made. Your LBAC credentials can be compared to a single security label for read access or your LBAC credentials compared to a single security label for write access. Updating and deleting are treated as being a read followed by a write. When an operation requires multiple comparisons to be made, each is made separately.

Which of your security labels is used

Even though you might hold multiple security labels only one is compared to the protecting security label. The label used is the one that meets these criteria:

- It is part of the security policy that is protecting the table being accessed.
- It was granted for the type of access (read or write).

If you do not have a security label that meets these criteria then a default security label is assumed that has empty values for all components.

How the comparison is made

Security labels are compared component by component. If a security label does not have a value for one of the components then an empty value is assumed. As each component is examined, the appropriate rules of the LBAC rule set are used to decide if the elements in your value for that component should be blocked by the elements in the value for the same component in the protecting label. If any of your values are blocked then your LBAC credentials are blocked by the protecting security label.

The LBAC rule set used in the comparison is designated in the security policy. To find out what the rules are and when each one is used, see the description of that rule set.

How exemptions affect comparisons

If you hold an exemption for the rule that is being used to compare two values then that comparison is not done and the protecting value is assumed not to block the value in your security label.

Example: The LBAC rule set is DB2LBACRULES and the security policy has two components. One component is of type ARRAY and the other is of type TREE. The user has been granted an exemption on the rule DB2LBACREADTREE, which is the rule used for read access when comparing values of components of type TREE. If the user attempts to read protected data then whatever value the user has for the TREE component, even if it is an empty value, will not block access because that rule is not used. Whether the user can read the data depends entirely on the values of the ARRAY component of the labels.

LBAC rule sets

An LBAC rule set is a predefined set of rules that are used when comparing security labels. When the values of a two security labels are being compared, one or more of the rules in the rule set will be used to determine if one value blocks another.

Each LBAC rule set is identified by a unique name. When you create a security policy you must specify the LBAC rule set that will be used with that policy. Any comparison of security labels that are part of that policy will use that LBAC rule set.

Each rule in a rule set is also identified by a unique name. You use the name of a rule when you are granting an exemption on that rule.

How many rules are in a set and when each rule is used can vary from rule set to rule set.

There is currently only one supported LBAC rule set. The name of that rule set is DB2LBACRULES.

LBAC rule set: DB2LBACRULES

The DB2LBACRULES LBAC rule set provides a traditional set of rules for comparing the values of security label components. It protects from both write-up and write-down.

What are write-up and write down?

Write-up and write-down apply only to components of type ARRAY and only to write access. Write up occurs when the value protecting data that you are writing to is higher than your value. Write-down is when the value protecting the data is lower than yours. By default neither write-up nor write-down is allowed, meaning that you can only write data that is protected by the same value that you have.

When comparing two values for the same component, which rules are used depends on the type of the component (ARRAY, SET, or TREE) and what type of access is being attempted (read, or write). This table lists the rules, tells when each is used, and describes how the rule determines if access is blocked.

Table 138. Summary of the DB2LBACRULES rules

Rule name	Used to compare values of this type of component	Used for this type of access	Access is blocked when this condition is met
DB2LBACREADARRAY	ARRAY	Read	The user's value is lower than the protecting value.
DB2LBACREADSET	SET	Read	There are one or more protecting values that the user does not hold.
DB2LBACREADTREE	TREE	Read	None of the user's values is equal to or an ancestor of one of the protecting values.
DB2LBACWRITEARRAY	ARRAY	Write	The user's value is higher than the protecting value or lower than the protecting value.[1]
DB2LBACWRITESET	SET	Write	There are one or more protecting values that the user does not hold.
DB2LBACWRITETREE	TREE	Write	None of the user's values is equal to or an ancestor of one of the protecting values.

Note:

1. The DB2LBACWRITEARRAY rule can be thought of as being two different rules combined. One prevents writing to data that is higher than your level (write-up) and the other prevents writing to data that is lower than your level (write-down). When granting an exemption to this rule you can exempt the user from either of these rules or from both.

How the rules handle empty values

All rules treat empty values the same way. An empty value blocks no other values and is blocked by any non-empty value.

DB2LBACREADSET and DB2LBACWRITESET examples

These examples are valid for a user trying to read or trying to write protected data. They assume that the values are for a component of type SET that has these elements: one two three four

Table 139. Examples of applying the DB2LBACREADSET and DB2LBACWRITESET rules.

User's value	Protecting value	Access blocked?
'one'	'one'	Not blocked. The values are the same.
'(one,two,three)'	'one'	Not blocked. The user's value contains the element 'one'.
'(one,two)'	'(one,two,four)'	Blocked. The element 'four' is in the protecting value but not in the user's value.
'()'	'one'	Blocked. An empty value is blocked by any non-empty value.
'one'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

DB2LBACREADTREE and DB2LBACWRITETREE

These examples are valid for both read access and write access. They assume that the values are for a component of type TREE that was defined in this way:

```
CREATE SECURITY LABEL COMPONENT mycomp
TREE (
    'Corporate'      ROOT,
    'Publishing'     UNDER 'Corporate',
    'Software'       UNDER 'Corporate',
    'Development'    UNDER 'Software',
    'Sales'          UNDER 'Software',
    'Support'        UNDER 'Software'
    'Business Sales' UNDER 'Sales'
    'Home Sales'     UNDER 'Sales'
)
```

This means the elements are in this arrangement:

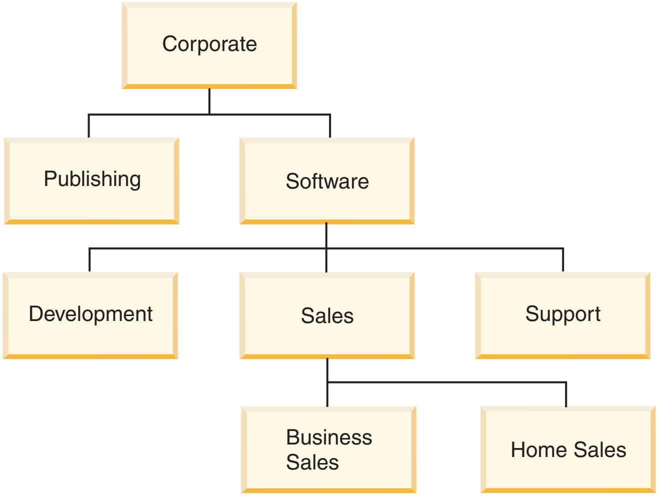


Table 140. Examples of applying the DB2LBACREADTREE and DB2LBACWRITETREE rules.

User's value	Protecting value	Access blocked?
'(Support,Sales)'	'Development'	Blocked. The element 'Development' is not one of the user's values and neither 'Support' nor 'Sales' is an ancestor of 'Development'.
'(Development,Software)'	'(Business Sales,Publishing)'	Not blocked. The element 'Software' is an ancestor of 'Business Sales'.
'(Publishing,Sales)'	'(Publishing,Support)'	Not blocked. The element 'Publishing' is in both sets of values.
'Corporate'	'Development'	Not blocked. The root value is an ancestor of all other values.
'()'	'Sales'	Blocked. An empty value is blocked by any non-empty value.
'Home Sales'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

DB2LBACREADARRAY examples

These examples are for read access only. They assume that the values are for a component of type ARRAY that includes these elements in this arrangement:

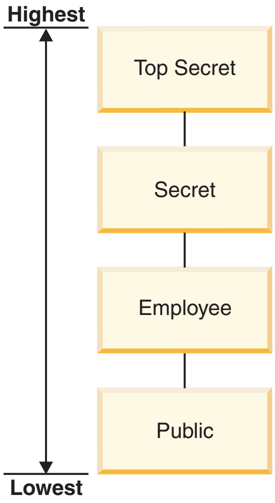


Table 141. Examples of applying the DB2LBACREADARRAY rule.

User's value	Protecting value	Read access blocked?
'Secret'	'Employee'	Not blocked. The element 'Secret' is higher than the element 'Employee'.
'Secret'	'Secret'	Not blocked. The values are the same.
'Secret'	'Top Secret'	Blocked. The element 'Top Secret' is higher than the element 'Secret'.
'()'	'Public'	Blocked. An empty value is blocked by any non-empty value.
'Public'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

DB2LBACWRITEARRAY examples

These examples are for write access only. They assume that the values are for a component of type ARRAY that includes these elements in this arrangement:

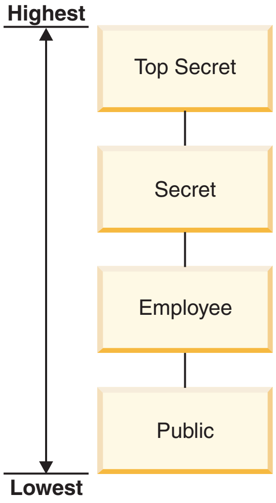


Table 142. Examples of applying the DB2LBACWRITEARRAY rule.

User's value	Protecting value	Write access blocked?
'Secret'	'Employee'	Blocked. The element 'Employee' is lower than the element 'Secret'.
'Secret'	'Secret'	Not blocked. The values are the same.
'Secret'	'Top Secret'	Blocked. The element 'Top Secret' is higher than the element 'Secret'.
'()'	'Public'	Blocked. An empty value is blocked by any non-empty value.
'Public'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

LBAC rule exemptions

When you hold an LBAC rule exemption on a particular rule of a particular security policy, that rule is not enforced when you try to access data protected by that security policy.

An exemption has no effect when comparing security labels of any security policy other than the one for which it was granted.

Example:

There are two tables: T1 and T2. T1 is protected by security policy P1 and T2 is protected by security policy P2. Both security policies have one component. The component of each is of type ARRAY. T1 and T2 each contain only one row of data. The security label that you hold for read access under security policy P1 does not allow you access to the row in T1. The security label that you hold for read access under security policy P2 does not allow you read access to the row in T2.

Now you are granted an exemption on DB2LBACREADARRAY under P1. You can now read the row from T1 but not the row from T2 because T2 is protected by a different security policy and you do not hold an exemption to the DB2LBACREADARRAY rule in that policy.

You can hold multiple exemptions. If you hold an exemption to every rule used by a security policy then you will have complete access to all data protected by that security policy.

Granting LBAC rule exemptions

You must be a security administrator to grant an LBAC rule exemption. To grant an LBAC rule exemption, use the SQL statement GRANT EXEMPTION ON RULE.

When you grant an LBAC rule exemption you provide this information:

- The rule or rules that the exemption is for
- The security policy that the exemption is for
- The user, group, or role to which you are granting the exemption

Important: LBAC rule exemptions provide very powerful access. Do not grant them without careful consideration.

Revoking LBAC rule exemptions

You must be a security administrator to revoke an LBAC rule exemption. To revoke an LBAC rule exemption, use the SQL statement REVOKE EXEMPTION ON RULE.

Determining the rule exemptions held by users

You can use the following query to determine the rule exemptions that are held by users:

```
SELECT A.grantee, A.accessrulename, B.secpolicyname
FROM syscat.securitypolicyexemptions A, syscat.securitypolicies B
WHERE A.secpolicyid = B.secpolicyid
```

Built-in functions for managing LBAC security labels

The built-in functions SECLABEL, SECLABEL_BY_NAME, and SECLABEL_TO_CHAR are provided for managing label-based access control (LBAC) security labels.

Each is described briefly here and in detail in the *SQL Reference*

SECLABEL

This built-in function is used to build a security label by specifying a security policy and values for each of the components in the label. The returned value has a data type of DB2SECURITYLABEL and is a security label that is part of the indicated security policy and has the indicated values for the components. It is not necessary that a security label with the indicated values already exists.

Example: Table T1 has two columns, the first has a data type of DB2SECURITYLABEL and the second has a data type of INTEGER. T1 is protected by security policy P1, which has three security label components: level, departments, and groups. If UNCLASSIFIED is an element of the component level, ALPHA and SIGMA are both elements of the component departments, and G2 is an element of the component groups then a security label could be inserted like this:

```
INSERT INTO T1 VALUES
   ( SECLABEL( 'P1', 'UNCLASSIFIED:(ALPHA,SIGMA):G2' ), 22 )
```

SECLABEL_BY_NAME

This built-in function accepts the name of a security policy and the name of a security label that is part of that security policy. It then returns the indicated security label as a DB2SECURITYLABEL. You must use this function when inserting an existing security label into a column that has a data type of DB2SECURITYLABEL.

Example: Table T1 has two columns, the first has a data type of DB2SECURITYLABEL and the second has a data type of INTEGER. The security label named L1 is part of security policy P1. This SQL inserts the security label:

```
INSERT INTO T1 VALUES ( SECLABEL_BY_NAME( 'P1', 'L1' ), 22 )
```

This SQL statement does not work:

```
INSERT INTO T1 VALUES ( P1.L1, 22 )     // Syntax Error!
```

SECLABEL_TO_CHAR

This built-in function returns a string representation of the values that make up a security label.

Example: Column C1 in table T1 has a data type of DB2SECURITYLABEL. T1 is protected by security policy P1, which has three security label components: level, departments, and groups. There is one row in T1 and the value in column C1 that has these elements for each of the components:

Component	Elements
level	SECRET
departments	DELTA and SIGMA
groups	G3

A user that has LBAC credentials that allow reading the row executes this SQL statement:

```
SELECT SECLABEL_TO_CHAR( 'P1', C1 ) AS C1 FROM T1
```

The output looks like this:

```
C1

'SECRET:(DELTA,SIGMA):G3'
```

Protection of data using LBAC

Label-based access control (LBAC) can be used to protect rows of data, columns of data, or both. Data in a table can only be protected by security labels that are part of the security policy protecting the table. Data protection, including adding a security policy, can be done when creating the table or later by altering the table.

You can add a security policy to a table and protect data in that table as part of the same CREATE TABLE or ALTER TABLE statement.

As a general rule you are not allowed to protect data in such a way that your current LBAC credentials do not allow you to write to that data.

Adding a security policy to a table

You can add a security policy to a table when you create the table by using the SECURITY POLICY clause of the CREATE TABLE statement. You can add a security policy to an existing table by using the ADD SECURITY POLICY clause of the ALTER TABLE statement. You do not need to have SECADM authority or have LBAC credentials to add a security policy to a table.

Security policies cannot be added to types of tables that cannot be protected by LBAC. See the overview of LBAC for a list of table types that cannot be protected by LBAC.

No more than one security policy can be added to any table.

Protecting rows

You can allow protected rows in a new table by including a column with a data type of DB2SECURITYLABEL when you create the table. The CREATE TABLE statement must also add a security policy to the table. You do not need to have SECADM authority or have any LBAC credentials to create such a table.

You can allow protected rows in an existing table by adding a column that has a data type of DB2SECURITYLABEL. To add such a column, either the table must already be protected by a security policy or the ALTER TABLE statement that adds the column must also add a security policy to the table. When the column is added, the security label you hold for write access is used to protect all existing rows. If you do not hold a security label for write access that is part of the security policy protecting the table then you cannot add a column that has a data type of DB2SECURITYLABEL.

After a table has a column of type DB2SECURITYLABEL you protect each new row of data by storing a security label in that column. The details of how this works are described in the topics about inserting and updating LBAC protected data. You must have LBAC credentials to insert rows into a table that has a column of type DB2SECURITYLABEL.

A column that has a data type of DB2SECURITYLABEL cannot be dropped and cannot be changed to any other data type.

Protecting columns

You can protect a column when you create the table by using the SECURED WITH column option of the CREATE TABLE statement. You can add protection to an existing column by using the SECURED WITH option in an ALTER TABLE statement.

To protect a column with a particular security label you must have LBAC credentials that allow you to write to data protected by that security label. You do not have to have SECADM authority.

Columns can only be protected by security labels that are part of the security policy protecting the table. You cannot protect columns in a table that has no security policy. You are allowed to protect a table with a security policy and protect one or more columns in the same statement.

You can protect any number of the columns in a table but a column can be protected by no more than one security label.

Reading LBAC protected data

When you try to read data protected by label-based access control (LBAC), your LBAC credentials for reading are compared to the security label that is protecting the data. If the protecting label does not block your credentials you are allowed to read the data.

In the case of a protected column the protecting security label is defined in the schema of the table. The protecting security label for that column is the same for every row in the table. In the case of a protected row the protecting security label is stored in the row in a column of type DB2SECURITYLABEL. It can be different for every row in the table.

The details of how your LBAC credentials are compared to a security label are given in the topic about how LBAC security labels are compared.

Reading protected columns

When you try to read from a protected column your LBAC credentials are compared with the security label protecting the column. Based on this comparison access will either be blocked or allowed. If access is blocked then an error is returned and the statement fails. Otherwise, the statement proceeds as usual.

Trying to read a column that your LBAC credentials do not allow you to read, causes the entire statement to fail.

Example:

Table T1 has two protected columns. The column C1 is protected by the security label L1. The column C2 is protected by the security label L2.

Assume that user Jyoti has LBAC credentials for reading that allow access to security label L1 but not to L2. If Jyoti issues the following SQL statement, the statement will fail:

```
SELECT * FROM T1
```

The statement fails because column C2 is included in the SELECT clause as part of the wildcard (*).

If Jyoti issues the following SQL statement it will succeed:

```
SELECT C1 FROM T1
```

The only protected column in the SELECT clause is C1, and Jyoti's LBAC credentials allow her to read that column.

Reading protected rows

If you do not have LBAC credentials that allow you to read a row it is as if that row does not exist for you.

When you read protected rows, only those rows to which your LBAC credentials allow read access are returned. This is true even if the column of type DB2SECURITYLABEL is not part of the SELECT clause.

Depending on their LBAC credentials, different users might see different rows in a table that has protected rows. For example, two users executing the statement `SELECT COUNT(*) FROM T1` may get different results if T1 has protected rows and the users have different LBAC credentials.

Your LBAC credentials affect not only SELECT statements but also other SQL statements like UPDATE, and DELETE. If you do not have LBAC credentials that allow you to read a row, you cannot affect that row.

Example:

Table T1 has these rows and columns. The column ROWSECURITYLABEL has a data type of DB2SECURITYLABEL.

Table 143. Example values in table T1

LASTNAME	DEPTNO	ROWSECURITYLABEL
Rjaibi	55	L2
Miller	77	L1
Fielding	11	L3
Bird	55	L2

Assume that user Dan has LBAC credentials that allow him to read data that is protected by security label L1 but not data protected by L2 or L3.

Dan issues the following SQL statement:

```
SELECT * FROM T1
```

The SELECT statement returns only the row for Miller. No error messages or warning are returned.

Dan's view of table T1 is this:

Table 144. Example values in view of table T1

LASTNAME	DEPTNO	ROWSECURITYLABEL
Miller	77	L1

The rows for Rjaibi, Fielding, and Bird are not returned because read access is blocked by their security labels. Dan cannot delete or update these rows. They will also not be included in any aggregate functions. For Dan it is as if those rows do not exist.

Dan issues this SQL statement:

```
SELECT COUNT(*) FROM T1
```

The statement returns a value of 1 because only the row for Miller can be read by the user Dan.

Reading protected rows that contain protected columns

Column access is checked before row access. If your LBAC credentials for read access are blocked by the security label protecting one of the columns you are selecting then the entire statement fails. If not, the statement continues and only the rows protected by security labels to which your LBAC credentials allow read access are returned.

Example

The column LASTNAME of table T1 is protected with the security label L1. The column DEPTNO is protected with security label L2. The column ROWSECURITYLABEL has a data type of DB2SECURITYLABEL. T1, including the data, looks like this:

Table 145. Example values in table T1

LASTNAME *Protected by L1*	**DEPTNO** *Protected by L2*	**ROWSECURITYLABEL**
Rjaibi	55	L2
Miller	77	L1
Fielding	11	L3

Assume that user Sakari has LBAC credentials that allow reading data protected by security label L1 but not L2 or L3.

Sakari issues this SQL statement:

```
SELECT * FROM T1
```

The statement fails because the SELECT clause uses the wildcard (*) which includes the column DEPTNO. The column DEPTNO is protected by security label L2, which Sakari's LBAC credentials do not allow her to read.

Sakari next issues this SQL statement:

```
SELECT LASTNAME, ROWSECURITYLABEL FROM T1
```

The select clause does not include any columns that Sakari is not able to read so the statement continues. Only one row is returned, however, because each of the other rows is protected by security label L2 or L3.

Table 146. Example output from query on table T1

LASTNAME	**ROWSECURITYLABEL**
Miller	L1

Inserting LBAC protected data

When you try to insert data into a protected column, or to insert a new row into a table with protected rows, your LBAC credentials determine how that INSERT statement is handled.

Inserting to protected columns

When you try to insert data into a protected column your LBAC credentials for writing are compared with the security label protecting that column. Based on this comparison access will either be blocked or allowed.

The details of how two security labels are compared are given in the topic about how LBAC security labels are compared.

If access is allowed, the statement proceeds as usual. If access is blocked, then the insert fails and an error is returned.

If you are inserting a row but do not provide a value for a protected column then a default value is inserted if one is available. This happens even if your LBAC credentials do not allow write access to that column. A default is available in the following cases:

- The column was declared with the WITH DEFAULT option

- The column is a generated column
- The column has a default value that is given through a BEFORE trigger
- The column has a data type of DB2SECURITYLABEL, in which case security label that you hold for write access is the default value

Inserting to protected rows

When you insert a new row into a table with protected rows, you do not have to provide a value for the column that is of type DB2SECURITYLABEL. If you do not provide a value for that column, the column is automatically populated with the security label you have been granted for write access. If you have not been granted a security label for write access, an error is returned and the insert fails.

By using built-in functions like SECLABEL, you can explicitly provide a security label to be inserted in a column of type DB2SECURITYLABEL. The provided security label is only used, however, if your LBAC credentials would allow you to write to data that is protected with the security label you are trying to insert.

If you provide a security label that you would not be able to write, then what happens depends on the security policy that is protecting the table. If the security policy has the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option, then the insert fails and an error is returned. If the security policy does not have the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option or if it instead has the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option, then the security label you provide is ignored and if you hold a security label for write access, it is used instead. If you do not hold a security label for write access, an error is returned.

Examples

Table T1 is protected by a security policy named P1 that was created without the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option. Table T1 has two columns but no rows. The columns are LASTNAME and LABEL. The column LABEL has a data type of DB2SECURITYLABEL.

User Joe holds a security label L2 for write access. Assume that the security label L2 allows him to write to data protected by security label L2 but not to data protected by security labels L1 or L3.

Joe issues the following SQL statement:

```
INSERT INTO T1 (LASTNAME, DEPTNO) VALUES ('Rjaibi', 11)
```

Because no security label was included in the INSERT statement, Joe's security label for write access is inserted into the LABEL row.

Table T1 now looks like this:

Table 147. Values in the example table T1 after first INSERT statement

LASTNAME	LABEL
Rjaibi	L2

Joe issues the following SQL statement, in which he explicitly provides the security label to be inserted into the column LABEL:

```
INSERT INTO T1 VALUES ('Miller', SECLABEL_BY_NAME('P1', 'L1') )
```

The SECLABEL_BY_NAME function in the statement returns a security label that is part of security policy P1 and is named L1. Joe is not allowed to write to data that is protected with L1 so he is not allowed to insert L1 into the column LABEL.

Because the security policy protecting T1 was created without the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option the security label that Joe holds for writing is inserted instead. No error or message is returned.

The table now looks like this:

Table 148. Values in example table T1 after second INSERT statement

LASTNAME	LABEL
Rjaibi	L2
Miller	L2

If the security policy protecting the table had been created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option then the insert would have failed and an error would have been returned.

Next Joe is granted an exemption to one of the LBAC rules. Assume that his new LBAC credentials allow him to write to data that is protected with security labels L1 and L2. The security label granted to Joe for write access does not change, it is still L2.

Joe issues the following SQL statement:

```
INSERT INTO T1 VALUES ('Bird', SECLABEL_BY_NAME('P1', 'L1') )
```

Because of his new LBAC credentials Joe is able to write to data that is protected by the security label L1. The insertion of L1 is therefore allowed. The table now looks like this:

Table 149. Values in example table T1 after third INSERT statement

LASTNAME	LABEL
Rjaibi	L2
Miller	L2
Bird	L1

Updating LBAC protected data

Your LBAC credentials must allow you write access to data before you can update it. In the case of updating a protected row, your LBAC credentials must also allow read access to the row.

Updating protected columns

When you try to update data in a protected column, your LBAC credentials are compared to the security label protecting the column. The comparison made is for write access. If write access is blocked then an error is returned and the statement fails, otherwise the update continues.

The details of how your LBAC credentials are compared to a security label are given in the topic about how LBAC security labels are compared.

Example:

Assume there is a table T1 in which column DEPTNO is protected by a security label L2 and column PAYSCALE is protected by a security label L3. T1, including its data, looks like this:

Table 150. Table T1

EMPNO	LASTNAME	**DEPTNO** *Protected by* **L2**	**PAYSCALE** *Protected by* **L3**
1	Rjaibi	11	4
2	Miller	11	7
3	Bird	11	9

User Lhakpa has no LBAC credentials. He issues this SQL statement:

```
UPDATE T1 SET EMPNO = 4
   WHERE LASTNAME = "Bird"
```

This statement executes without error because it does not update any protected columns. T1 now looks like this:

Table 151. Table T1 After Update

EMPNO	LASTNAME	**DEPTNO** *Protected by* **L2**	**PAYSCALE** *Protected by* **L3**
1	Rjaibi	11	4
2	Miller	11	7
4	Bird	11	9

Lhakpa next issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 55
   WHERE LASTNAME = "Miller"
```

This statement fails and an error is returned because DEPTNO is protected and Lhakpa has no LBAC credentials.

Assume Lhakpa is granted LBAC credentials and that allow the access summarized in the following table. The details of what those credentials are and what elements are in the security labels are not important for this example.

Security label protecting the data	Can read?	Can Write?
L2	No	Yes
L3	No	No

Lhakpa issues this SQL statement again:

```
UPDATE T1 SET DEPTNO = 55
   WHERE LASTNAME = "Miller"
```

This time the statement executes without error because Lhakpa's LBAC credentials allow him to write to data protected by the security label that is protecting the column DEPTNO. It does not matter that he is not able to read from that same column. The data in T1 now looks like this:

Table 152. Table T1 After Second Update

EMPNO	LASTNAME	**DEPTNO** *Protected by* **L2**	**PAYSCALE** *Protected by* **L3**
1	Rjaibi	11	4
2	Miller	55	7
4	Bird	11	9

Next Lhakpa issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 55, PAYSCALE = 4
   WHERE LASTNAME = "Bird"
```

The column PAYSCALE is protected by the security label L3 and Lhakpa's LBAC credentials do not allow him to write to it. Because Lhakpa is unable to write to the column, the update fails and no data is changed.

Updating protected rows

If your LBAC credentials do not allow you to read a row, then it is as if that row does not exist for you so there is no way for you to update that row. For rows that you are able to read, you must also be able to write to the row in order to update it.

When you try to update a row, your LBAC credentials for writing are compared to the security label protecting the row. If write access is blocked, the update fails and an error is returned. If write access is not blocked, then the update continues.

The update that is performed is done the same way as an update to a non-protected row except for the treatment of the column that has a data type of DB2SECURITYLABEL. If you do not explicitly set the value of that column, it is automatically set to the security label that you hold for write access. If you do not have a security label for write access, an error is returned and the statement fails.

If the update explicitly sets the column that has a data type of DB2SECURITYLABEL, then your LBAC credentials are checked again. If the update you are trying to perform would create a row that your current LBAC credentials would not allow you to write to, then what happens depends on the security policy that is protecting the table. If the security policy has the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option, then the update fails and an error is returned. If the security policy does not have the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option or if it instead has the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option, then the security label you provide is ignored and if you hold a security label for write access, it is used instead. If you do not hold a security label for write access, an error is returned.

Example:

Assume that table T1 is protected by a security policy named P1 and has a column named LABEL that has a data type of DB2SECURITYLABEL.

T1, including its data, looks like this:

Table 153. Table T1

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	11	L1
2	Miller	11	L2
3	Bird	11	L3

Assume that user Jenni has LBAC credentials that allow her to read and write data protected by the security labels L0 and L1 but not data protected by any other security labels. The security label she holds for both read and write is L0. The details of her full credentials and of what elements are in the labels are not important for this example.

Jenni issues this SQL statement:

```
SELECT * FROM T1
```

Jenni sees only one row in the table:

Table 154. Jenni's SELECT Query Result

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	11	L1

The rows protected by labels L2 and L3 are not included in the result set because Jenni's LBAC credentials do not allow her to read those rows. For Jenni it is as if those rows do not exist.

Jenni issues these SQL statements:

```
UPDATE T1 SET DEPTNO = 44 WHERE DEPTNO = 11;
SELECT * FROM T1;
```

The result set returned by the query looks like this:

Table 155. Jenni's UPDATE & SELECT Query Result

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L0

The actual data in the table looks like this:

Table 156. Table T1

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L0
2	Miller	11	L2
3	Bird	11	L3

The statement executed without error but affected only the first row. The second and third rows are not readable by Jenni so they are not selected for update by the statement even though they meet the condition in the WHERE clause.

Notice that the value of the LABEL column in the updated row has changed even though that column was not explicitly set in the UPDATE statement. The column was set to the security label that Jenni held for writing.

Now Jenni is granted LBAC credentials that allow her to read data protected by any security label. Her LBAC credentials for writing do not change. She is still only able to write to data protected by L0 and L1.

Jenni again issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 44 WHERE DEPTNO = 11
```

This time the update fails because of the second and third rows. Jenni is able to read those rows, so they are selected for update by the statement. She is not, however, able to write to them because they are protected by security labels L2 and L3. The update does not occur and an error is returned.

Jenni now issues this SQL statement:

```
UPDATE T1
 SET DEPTNO = 55, LABEL = SECLABEL_BY_NAME( 'P1', 'L2' )
 WHERE LASTNAME = "Rjaibi"
```

The SECLABEL_BY_NAME function in the statement returns the security label named L2. Jenni is trying to explicitly set the security label protecting the first row. Jenni's LBAC credentials allow her to read the first row, so it is selected for update. Her LBAC credentials allow her to write to rows protected by the security label L0 so she is allowed to update the row. Her LBAC credentials would not, however, allow her to write to a row protected by the security label L2, so she is not allowed to set the column LABEL to that value. The statement fails and an error is returned. No columns in the row are updated.

Jenni now issues this SQL statement:

```
UPDATE T1 SET LABEL = SECLABEL_BY_NAME( 'P1', 'L1' ) WHERE LASTNAME = "Rjaibi"
```

The statement succeeds because she would be able to write to a row protected by the security label L1.

T1 now looks like this:

Table 157. Table T1

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L1
2	Miller	11	L2
3	Bird	11	L3

Updating protected rows that contain protected columns

If you try to update protected columns in a table with protected rows then your LBAC credentials must allow writing to of all of the protected columns affected by the update, otherwise the update fails and an error is returned. This is as described in section about updating protected columns, earlier. If you are allowed to update all of the protected columns affected by the update you will still only be able to update rows that your LBAC credentials allow you to both read from and write to. This is as described in the section about updating protected rows, earlier. The handling of a column with a data type of DB2SECURITYLABEL is the same whether the update affects protected columns or not.

If the column that has a data type of DB2SECURITYLABEL is itself a protected column then your LBAC credentials must allow you to write to that column or you cannot update any of the rows in the table.

Deleting or dropping LBAC protected data

Your ability to delete data in tables protected by LBAC depend on your LBAC credentials.

Deleting protected rows

If your LBAC credentials do not allow you to read a row, it is as if that row does not exist for you so there is no way for you to delete it. To delete a row that you are able to read, your LBAC credentials must also allow you to write to the row. To delete any row in a table that has protected columns you must have LBAC credentials that allow you to write to all protected columns in the table.

When you try to delete a row, your LBAC credentials for writing are compared to the security label protecting the row. If the protecting security label blocks write access by your LBAC credentials, the DELETE statement fails, an error is returned, and no rows are deleted.

Example

Protected table T1 has these rows:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2
Fielding	77	L3

Assume that user Pat has LBAC credentials such that her access is as summarized in this table:

Security label	Read access?	Write access?
L1	Yes	Yes
L2	Yes	No
L3	No	No

The exact details of her LBAC credentials and of the security labels are unimportant for this example.

Pat issues the following SQL statement:

```
SELECT * FROM T1 WHERE DEPTNO != 999
```

The statement executes and returns this result set:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Miller	77	L1

LASTNAME	DEPTNO	LABEL
Bird	55	L2

The last row of T1 is not included in the results because Pat does not have read access to that row. It is as if that row does not exist for Pat.

Pat issues this SQL statement:

```
DELETE FROM T1 WHERE DEPTNO != 999
```

Pat does not have write access to the first or third row, both of which are protected by L2. So even though she can read the rows she cannot delete them. The DELETE statement fails and no rows are deleted.

Pat issues this SQL statement:

```
DELETE FROM T1 WHERE DEPTNO = 77;
```

This statement succeeds because Pat is able to write to the row with Miller in the LASTNAME column. That is the only row selected by the statement. The row with Fielding in the LASTNAME column is not selected because Pat's LBAC credentials do not allow her to read that row. That row is never considered for the delete so no error occurs.

The actual rows of the table now look like this:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Bird	55	L2
Fielding	77	L3

Deleting rows that have protected columns

To delete any row in a table that has protected columns you must have LBAC credentials that allow you to write to all protected columns in the table. If there is any row in the table that your LBAC credentials do not allow you to write to then the delete will fail and an error will be returned.

If the table has both protected columns and protected rows then to delete a particular row you must have LBAC credentials that allow you to write to every protected column in the table and also to read from and write to the row that you want to delete.

Example

In protected table T1, the column DEPTNO is protected by the security label L2. T1 contains these rows:

LASTNAME	DEPTNO *Protected by* **L2**	LABEL
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2
Fielding	77	L3

Assume that user Benny has LBAC credentials that allow him the access summarized in this table:

Security label	Read access?	Write access?
L1	Yes	Yes
L2	Yes	No
L3	No	No

The exact details of his LBAC credentials and of the security labels are unimportant for this example.

Benny issues the following SQL statement:

```
DELETE FROM T1 WHERE DEPTNO = 77
```

The statement fails because Benny does not have write access to the column DEPTNO.

Now Benny's LBAC credentials are changed so that he has access as summarized in this table:

Security label	Read access?	Write access?
L1	Yes	Yes
L2	Yes	Yes
L3	Yes	No

Benny issues this SQL statement again:

```
DELETE FROM T1 WHERE DEPTNO = 77
```

This time Benny has write access to the column DEPTNO so the delete continues. The delete statement selects only the row that has a value of Miller in the LASTNAME column. The row that has a value of Fielding in the LASTNAME column is not selected because Benny's LBAC credentials do not allow him to read that row. Because the row is not selected for deletion by the statement it does not matter that Benny is unable to write to the row.

The one row selected is protected by the security label L1. Benny's LBAC credentials allow him to write to data protected by L1 so the delete is successful.

The actual rows in table T1 now look like this:

LASTNAME	DEPTNO *Protected by* **L2**	LABEL
Rjaibi	55	L2
Bird	55	L2
Fielding	77	L3

Dropping protected data

You cannot drop a column that is protected by a security label unless your LBAC credentials allow you to write to that column.

A column with a data type of DB2SECURITYLABEL cannot be dropped from a table. To remove it you must first drop the security policy from the table. When you drop the security policy the table is no longer protected with LBAC and the data type of the column is automatically changed from DB2SECURITYLABEL to VARCHAR(128) FOR BIT DATA. The column can then be dropped.

Your LBAC credentials do not prevent you from dropping entire tables or databases that contain protected data. If you would normally have permission to drop a table or a database you do not need any LBAC credentials to do so, even if the database contains protected data.

Removing LBAC protection from data

You must have SECADM authority to remove the security policy from a table. To remove the security policy from a table you use the DROP SECURITY POLICY clause of the ALTER TABLE statement. This also automatically removes protection from all rows and all columns of the table.

Removing protection from rows

In a table that has protected rows every row must be protected by a security label. There is no way to remove LBAC protection from individual rows.

A column of type DB2SECURITYLABEL cannot be altered or removed except by removing the security policy from the table.

Removing protection from columns

Protection of a column can be removed using the DROP COLUMN SECURITY clause of the SQL statement ALTER TABLE. To remove the protection from a column you must have LBAC credentials that allow you to read from and write to that column in addition to the normal privileges and authorities needed to alter a table.

Chapter 62. DB2 audit facility

To manage access to your sensitive data, you can use a variety of authentication and access control mechanisms to establish rules and controls for acceptable data access. But to protect against and discover unknown or unacceptable behaviors you can monitor data access by using the DB2 audit facility.

Successful monitoring of unwanted data access and subsequent analysis can lead to improvements in the control of data access and the ultimate prevention of malicious or careless unauthorized access to data. The monitoring of application and individual user access, including system administration actions, can provide a historical record of activity on your database systems.

The DB2 audit facility generates, and allows you to maintain, an audit trail for a series of predefined database events. The records generated from this facility are kept in an audit log file. The analysis of these records can reveal usage patterns that would identify system misuse. Once identified, actions can be taken to reduce or eliminate such system misuse.

The audit facility provides the ability to audit at both the instance and the individual database level, independently recording all instance and database level activities with separate logs for each. The system administrator (who holds SYSADM authority) can use the **db2audit** tool to configure audit at the instance level as well as to control when such audit information is collected. The system administrator can use the **db2audit** tool to archive both instance and database audit logs as well as to extract audit data from archived logs of either type.

The security administrator (who holds SECADM authority within a database) can use audit policies in conjunction with the SQL statement, AUDIT, to configure and control the audit requirements for an individual database. The security administrator can use the following audit routines to perform the specified tasks:

- The SYSPROC.AUDIT_ARCHIVE stored procedure archives audit logs.
- The SYSPROC.AUDIT_LIST_LOGS table function allows you to locate logs of interest.
- The SYSPROC.AUDIT_DELIM_EXTRACT stored procedure extracts data into delimited files for analysis.

The security administrator can grant EXECUTE privilege on these routines to another user, therefore enabling the security administrator to delegate these tasks, if required.

When working in a partitioned database environment, many of the auditable events occur at the database partition at which the user is connected (the coordinator partition) or at the catalog partition (if they are not the same database partition). The implication of this is that audit records can be generated by more than one database partition. Part of each audit record contains information identifying the coordinator partition and originating partition (the partition where audit record originated).

At the instance level, the audit facility must be stopped and started explicitly by use of the **db2audit start** and **db2audit stop** commands. When you start instance-level auditing, the audit facility uses existing audit configuration information. Since the audit facility is independent of the DB2 database server, it will remain active even if the instance is stopped. In fact, when the instance is stopped, an audit record may be generated in the audit log. To start auditing at the database level, first you need to create an audit policy, then you associate this audit policy with the objects you want to monitor, such as, authorization IDs, database authorities, trusted contexts or particular tables.

Categories of audit records

There are different categories of audit records that may be generated. In the following description of the categories of events available for auditing, you should notice that following the name of each category is a one-word keyword used to identify the category type. The categories of events available for auditing are:

- Audit (AUDIT). Generates records when audit settings are changed or when the audit log is accessed.
- Authorization Checking (CHECKING). Generates records during authorization checking of attempts to access or manipulate DB2 database objects or functions.
- Object Maintenance (OBJMAINT). Generates records when creating or dropping data objects, and when altering certain objects.
- Security Maintenance (SECMAINT). Generates records when:
  - Granting or revoking object privileges or database authorities
  - Granting or revoking security labels or exemptions
  - Altering the group authorization, role authorization, or override or restrict attributes of an LBAC security policy
  - Granting or revoking the SETSESSIONUSER privilege
  - Modifying any of the SYSADM_GROUP, SYSCTRL_GROUP, SYSMAINT_GROUP, or SYSMON_GROUP configuration parameters.
- System Administration (SYSADMIN). Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.
- User Validation (VALIDATE). Generates records when authenticating users or retrieving system security information.
- Operation Context (CONTEXT). Generates records to show the operation context when a database operation is performed. This category allows for better interpretation of the audit log file. When used with the log's event correlator field, a group of events can be associated back to a single database operation. For example, a query statement for dynamic queries, a package identifier for static queries, or an indicator of the type of operation being performed, such as CONNECT, can provide needed context when analyzing audit results.

  Note: The SQL or XQuery statement providing the operation context might be very long and is completely shown within the CONTEXT record. This can make the CONTEXT record very large.
- Execute (EXECUTE). Generates records during the execution of SQL statements.

For any of the categories listed previously, you can audit failures, successes, or both.

Any operations on the database server may generate several records. The actual number of records generated in the audit log depends on the number of categories of events to be recorded as specified by the audit facility configuration. It also depends on whether successes, failures, or both, are audited. For this reason, it is important to be selective of the events to audit.

Audit policies

The security administrator can use audit policies to configure the audit facility to gather information only about the data and objects that are needed.

The security administrator can create audit policies to control what is audited within an individual database. The following objects can have an audit policy associated with them:

- The whole database

  All auditable events that occur within the database are audited according to the audit policy.
- Tables

  All data manipulation language (DML) and XQUERY access to the table (untyped), MQT (materialized query table), or nickname is audited. Only EXECUTE category audit events with or without data are generated when the table is accessed even if the policy indicates that other categories should be audited.
- Trusted contexts

  All auditable events that happen within a trusted connection defined by the particular trusted context are audited according to the audit policy.
- Authorization IDs representing users, groups, or roles

  All auditable events that are initiated by the specified user are audited according to the audit policy.

  All auditable events that are initiated by users that are a member of the group or role are audited according to the audit policy. Indirect role membership, such as through other roles or groups, is also included.

  You can capture similar data by using the Work Load Management event monitors by defining a work load for a group and capturing the activity details. You should be aware that the mapping to workloads can involve attributes in addition to just the authorization ID, which can cause you to not achieve the wanted granularity in auditing, or if those other attributes are modified, connections may map to different (possibly unmonitored) workloads. The auditing solution provides a guarantee that a user, group or role will be audited.
- Authorities (SYSADM, SECADM, DBADM, SQLADM, WLMADM, ACCESSCTRL, DATAACCESS, SYSCTRL, SYSMAINT, SYSMON)

  All auditable events that are initiated by a user that holds the specified authority, even if that authority is unnecessary for the event, are audited according to the audit policy.

The security administrator can create multiple audit policies. For example, your company might want a policy for auditing sensitive data and a policy for auditing the activity of users holding DBADM authority. If multiple audit policies are in effect for a statement, all events required to be audited by each of the audit policies are audited (but audited only once). For example, if the database's audit policy requires auditing successful EXECUTE events for a particular table and the user's audit policy requires auditing failures of EXECUTE events for that same table, both successful and failed attempts at accessing that table are audited.

For a specific object, there can only be one audit policy in effect. For example, you cannot have multiple audit policies associated with the same table at the same time.

An audit policy cannot be associated with a view or a typed table. Views that access a table that has an associated audit policy are audited according to the underlying table's policy.

The audit policy that applies to a table does not automatically apply to a MQT based on that table. If you associate an audit policy with a table, associate the same policy with any MQT based on that table.

Auditing performed during a transaction is done based on the audit policies and their associations at the start of the transaction. For example, if the security administrator associates an audit policy with a user and that user is in a transaction at the time, the audit policy does not affect any remaining statements performed within that transaction. Also, changes to an audit policy do not take effect until they are committed. If the security administrator issues an ALTER AUDIT POLICY statement, it does not take effect until the statement is committed.

The security administrator uses the CREATE AUDIT POLICY statement to create an audit policy, and the ALTER AUDIT POLICY statement to modify an audit policy. These statements can specify:

- The status values for events to be audited: `None`, `Success`, `Failure`, or `Both`.

  Only auditable events that match the specified status value are audited.
- The server behavior when errors occur during auditing.

The security administrator uses the AUDIT statement to associate an audit policy with the current database or with a database object, at the current server. Any time the object is in use, it is audited according to this audit policy.

To delete an audit policy, the security administrator uses the DROP statement. You cannot drop an audit policy if it is associated with any object. Use the AUDIT REMOVE statement to remove any remaining association with an object. To add metadata to an audit policy, the security administrator uses the COMMENT statement.

Events generated before a full connection has been established

For some events generated during connect and a switch user operation, the only audit policy information available is the policy that is associated with the database. These events are shown in the following table:

Table 158. Connection events

Event	Audit category	Comment
CONNECT	CONTEXT	
CONNECT_RESET	CONTEXT	
AUTHENTICATION	VALIDATE	This includes authentication during both connect and switch user within a trusted connection.
CHECKING_FUNC	CHECKING	The access attempted is SWITCH_USER.

These events are audited based only on the audit policy associated with the database and not with audit policies associated with any other object such as a user, their groups, or authorities. For the CONNECT and AUTHENTICATION events that occur during connect, the instance-level audit settings are used until the database is activated. The database is activated either during the first connection or when the ACTIVATE DATABASE command is issued.

Effect of switching user

If a user is switched within a trusted connection, no remnants of the original user are left behind. In this case, the audit policies associated with the original user are no longer considered, and the applicable audit policies are re-evaluated according to the new user. Any audit policy associated with the trusted connection is still in effect.

If a SET SESSION USER statement is used, only the session authorization ID is switched. The audit policy of the authorization ID of the original user (the system authorization ID) remains in effect and the audit policy of the new user is used as well. If multiple SET SESSION USER statements are issued within a session, only the audit policies associated with the original user (the system authorization ID) and the current user (the session authorization ID) are considered.

Data definition language restrictions

The following data definition language (DDL) statements are called AUDIT exclusive SQL statements:

- AUDIT
- CREATE AUDIT POLICY, ALTER AUDIT POLICY, and DROP AUDIT POLICY
- DROP ROLE and DROP TRUSTED CONTEXT, if the role or trusted context being dropped is associated with an audit policy

AUDIT exclusive SQL statements have some restrictions in their use:

- Each statement must be followed by a COMMIT or ROLLBACK.
- These statements cannot be issued within a global transaction, for example an XA transaction.

Only one uncommitted AUDIT exclusive DDL statement is allowed at a time across all partitions. If an uncommitted AUDIT exclusive DDL statement is executing, subsequent AUDIT exclusive DDL statements wait until the current AUDIT exclusive DDL statement commits or rolls back.

Note: Changes are written to the catalog, but do not take effect until COMMIT, even for the connection that issues the statement.

Example of auditing any access to a specific table

Consider a company where the EMPLOYEE table contains extremely sensitive information and the company wants to audit any and all SQL access to the data in that table. The EXECUTE category can be used to track all access to a table; it audits the SQL statement, and optionally the input data value provided at execution time for that statement.

There are two steps to track activity on the table. First, the security administrator creates an audit policy that specifies the EXECUTE category, and then the security administrator associates that policy with the table:

```
CREATE AUDIT POLICY SENSITIVEDATAPOLICY
    CATEGORIES EXECUTE STATUS BOTH ERROR TYPE AUDIT
COMMIT

AUDIT TABLE EMPLOYEE USING POLICY SENSITIVEDATAPOLICY
COMMIT
```

Example of auditing any actions by SYSADM or DBADM

In order to complete their security compliance certification, a company must show that any and all activities within the database by those people holding system administration (SYSADM) or database administrative (DBADM) authority can be monitored.

To capture all actions within the database, both the EXECUTE and SYSADMIN categories should be audited. The security administrator creates an audit policy that audits these two categories. The security administrator can use the AUDIT statement to associate this audit policy with the SYSADM and DBADM authorities. Any user that holds either SYSADM or DBADM authority will then have any auditable events logged. The following example shows how to create such an audit policy and associate it with the SYSADM and DBADM authorities:

```
CREATE AUDIT POLICY ADMINSPOLICY CATEGORIES EXECUTE STATUS BOTH,
   SYSADMIN STATUS BOTH ERROR TYPE AUDIT
COMMIT
AUDIT SYSADM, DBADM USING POLICY ADMINSPOLICY
COMMIT
```

Example of auditing any access by a specific role

A company has allowed its web applications access to their corporate database. The exact individuals using the web applications are unknown. Only the role that is used is known and that role is used to manage the database authorizations. The company wants to monitor the actions of anyone who is a member of that role in order to examine the requests they are submitting to the database and to ensure that they only access the database through the web applications.

The EXECUTE category contains the necessary level of auditing to track the activity of the users for this situation. The first step is to create the appropriate audit policy and associate it with the roles that are used by the web applications (in this example, the roles are TELLER and CLERK):

```
CREATE AUDIT POLICY WEBAPPPOLICY CATEGORIES EXECUTE WITH DATA
   STATUS BOTH ERROR TYPE AUDIT
COMMIT
AUDIT ROLE TELLER, ROLE CLERK USING POLICY WEBAPPPOLICY
COMMIT
```

Example of enabling auditing for a database

A company wants to determine who is making DDL changes (example: ALTER TABLE) on the database named SAMPLE.

```
CONNECT TO SAMPLE

CREATE AUDIT POLICY ALTPOLICY CATEGORIES AUDIT STATUS BOTH,
    OBJMAINT STATUS BOTH,  CHECKING STATUS BOTH,
    EXECUTE STATUS BOTH, ERROR TYPE NORMAL

AUDIT DATABASE USING POLICY ALTPOLICY
```

Storage and analysis of audit logs

Archiving the audit log moves the active audit log to an archive directory while the server begins writing to a new, active audit log. Later, you can extract data from the archived log into delimited files and then load data from these files into DB2 database tables for analysis.

Configuring the location of the audit logs allows you to place the audit logs on a large, high-speed disk, with the option of having separate disks for each member in a multiple member environment, such as a DB2 pureScale environment or a partitioned database environment. In a multiple member environment, the path for the active audit log can be a directory that is unique to each member. Having a unique directory for each member helps to avoid file contention, because each member is writing to a different disk.

The default path for the audit logs on Windows operating systems is *instance*\security\auditdata and on Linux and UNIX operating systems is *instance*/security/auditdata. If you do not want to use the default location, you can choose different directories (you can create new directories on your system to use as alternative locations, if they do not already exist). To set the path for the active audit log location and the archived audit log location, use the **db2audit configure** command with the **datapath** and **archivepath** parameters, as shown in this example:

```
db2audit configure datapath /auditlog archivepath /auditarchive
```

The audit log storage locations you set using **db2audit** apply to all databases in the instance.

Note: If there are multiple instances on the server, then each instance should each have separate data and archive paths.

The path for active audit logs (datapath) in a multiple member environment

In a multiple member environment, the same active audit log location (set by the **datapath** parameter) must be used on each member. There are two ways to accomplish this:

1. Use database member expressions when you specify the **datapath** parameter. Using database member expressions allows the member number to be included in the path of the audit log files and results in a different path on each database member.
2. Use a shared drive that is the same on all members.

You can use database member expressions anywhere within the value you specify for the **datapath** parameter. For example, on a three member system, where the database member number is 10, the following command:

```
db2audit configure datapath '/pathForNode $N'
```

uses the following paths:

- /pathForMember10
- /pathForMember20
- /pathForMember30

Note: You cannot use database member expressions to specify the archive log file path (**archivepath** parameter).

Archiving active audit logs

The system administrator can use the **db2audit** tool to archive both instance and database audit logs as well as to extract audit data from archived logs of either type.

The security administrator, or a user to whom the security administrator has granted EXECUTE privilege on the audit routines, can archive the active audit log by running the SYSPROC.AUDIT_ARCHIVE stored procedure. To extract data from the log and load it into delimited files, they can use the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure.

These are the steps to archive and extract the audit logs using the audit routines:
1. Schedule an application to perform regular archives of the active audit log using the stored procedure SYSPROC.AUDIT_ARCHIVE.
2. Determine which archived log files are of interest. Use the SYSPROC.AUDIT_LIST_LOGS table function to list all of the archived audit logs.
3. Pass the file name as a parameter to the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure to extract data from the log and load it into delimited files.
4. Load the audit data into DB2 database tables for analysis.

The archived log files do not need to be immediately loaded into tables for analysis; they can be saved for future analysis. For example, they may only need to be looked at when a corporate audit is taking place.

If a problem occurs during archive, such as running out of disk space in the archive path, or the archive path does not exist, the archive process fails and an interim log file with the file extension `.bk` is generated in the audit log data path, for example, `db2audit.instance.log.0.20070508172043640941.bk`. After the problem is resolved (by allocating sufficient disk space in the archive path, or by creating the archive path) you must move this interim log to the archive path. Then, you can treat it in the same way as a successfully archived log.

Archiving active audit logs in a multiple member environment

In a multiple member environment, if the archive command is issued while the instance is running, the archive process automatically runs on every member. The same timestamp is used in the archived log file name on all members. For example, on a three member system, where the database member number is 10, the following command:

```
db2audit archive to /auditarchive
```

creates the following files:
- `/auditarchive/db2audit.log.10.`*`timestamp`*
- `/auditarchive/db2audit.log.20.`*`timestamp`*
- `/auditarchive/db2audit.log.30.`*`timestamp`*

If the archive command is issued while the instance is not running, you can control on which member the archive is run by one of the following methods:

- Use the **node** option with the **db2audit** command to perform the archive for the current member only.
- Use the **db2_all** command to run the archive on all members.

  For example:

  ```
  db2_all db2audit archive node to /auditarchive
  ```

  This sets the **DB2NODE** environment variable to indicate on which members the command is invoked.

Alternatively, you can issue an individual archive command on each member separately. For example:

- On member 10:

  ```
  db2audit archive node 10 to /auditarchive
  ```
- On member 20:

  ```
  db2audit archive node 20 to /auditarchive
  ```
- On member 30:

  ```
  db2audit archive node 30 to /auditarchive
  ```

Note: When the instance is not running, the timestamps in the archived audit log file names are not the same on each member.

Note: It is recommended that the archive path is shared across all members, but it is not required.

Note: The AUDIT_DELIM_EXTRACT stored procedure and AUDIT_LIST_LOGS table function can only access the archived log files that are visible from the current (coordinator) member.

Example of archiving a log and extracting data to a table

To ensure their audit data is captured and stored for future use, a company needs to create a new audit log every six hours and archive the current audit log to a WORM drive. The company schedules the following call to the SYSPROC.AUDIT_ARCHIVE stored procedure to be issued every six hours by the security administrator, or by a user to whom the security administrator has granted EXECUTE privilege on the AUDIT_ARCHIVE stored procedure. The path to the archived log is the default archive path, `/auditarchive`, and the archive runs on all members:

```
CALL SYSPROC.AUDIT_ARCHIVE( '/auditarchive', -2 )
```

As part of their security procedures, the company has identified and defined a number of suspicious behaviors or disallowed activities that it needs to watch for in the audit data. They want to extract all the data from the one or more audit logs, place it in a relational table, and then use SQL queries to look for these activities. The company has decided on appropriate categories to audit and has associated the necessary audit policies with the database or other database objects.

For example, they can call the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure to extract the archived audit logs for all categories from all members that were created with a timestamp in April 2006, using the default delimiter:

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT(
     '', '', '/auditarchive',  'db2audit.%.200604%', '' )
```

In another example, they can call the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure to extract the archived audit records with success events from the EXECUTE category and failure events from the CHECKING category, from a file with the timestamp they are interested in:

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT( '', '', '/auditarchive',
    'db2audit.%.20060419034937', 'category
     execute status success, checking status failure );
```

The EXECUTE category for auditing SQL statements

Use the EXECUTE category to accurately track the SQL statements that are issued by a user. In Version 9.5 and earlier releases, you had to use the CONTEXT category to find this information.

As part of a comprehensive security policy, a company can require the ability to retroactively go back a set number of years and analyze the effects of any particular request against certain tables in their database. To do this, a company must institute a policy of archiving their weekly backups and associated log files such that they can reconstitute the database for any chosen moment in time. Also required, is sufficient database audit information captured about every request made against the database to allow, at any future time, the replay and analysis of any request against the relevant, restored database. This requirement can cover both static and dynamic SQL statements.

This EXECUTE category captures the SQL statement text as well as the compilation environment and other values that are needed to replay the statement at a later date. For example, replaying the statement can show you exactly which rows a SELECT statement returned. In order to re-run a statement, the database tables must first be restored to their state when the statement was issued.

When you audit using the EXECUTE category, the statement text for both static and dynamic SQL is recorded, as are input parameter markers and host variables. You can configure the EXECUTE category to be audited with or without input values.

Note: Global variables are not audited.

The auditing of EXECUTE events takes place at the completion of the event (for SELECT statements this is on cursor close). The status that the event completed with is also stored. Because EXECUTE events are audited at completion, long-running queries do not immediately appear in the audit log.

Note: The preparation of a statement is not considered part of the execution. Most authorization checks are performed at prepare time (for example, SELECT privilege). This means that statements that fail during prepare due to authorization errors do not generate EXECUTE events.

Statement Value Index, Statement Value Type and Statement Value Data fields may be repeated for a given execute record. For the report format generated by the extraction, each record lists multiple values. For the delimited file format, multiple rows are used. The first row has an event type of STATEMENT and no values. Following rows have an event type of DATA, with one row for each data value associated with the SQL statement. You can use the event correlator and application ID fields to link STATEMENT and DATA rows together. The columns Statement Text, Statement Isolation Level, and Compilation Environment Description are not present in the DATA events.

The statement text and input data values that are audited are converted into the database code page when they are stored on disk (all audited fields are stored in the database code page). No error is returned if the code page of the input data is not compatible with the database code page; the unconverted data will be logged instead. Because each database has it's own audit log, databases having different code pages does not cause a problem.

ROLLBACK and COMMIT are audited when executed by the application, and also when issued implicitly as part of another command, such as BIND.

After an EXECUTE event has been audited due to access to an audited table, all statements that affect which other statements are executed within a unit of work, are audited. These statements are COMMIT, ROLLBACK, ROLLBACK TO SAVEPOINT and SAVEPOINT.

Savepoint ID field

You can use the Savepoint ID field to track which statements were affected by a ROLLBACK TO SAVEPOINT statement. An ordinary DML statement (such as SELECT, INSERT, and so on) has the current savepoint ID audited. However, for the ROLLBACK TO SAVEPOINT statement, the savepoint ID that is rolled back to will be audited instead. Therefore, every statement with a savepoint ID greater than or equal to that ID will be rolled back, as demonstrated by the following example. The table shows the sequence of statements run; all events with a Savepoint ID greater than or equal to 2 will be rolled back. Only the value of 3 (from the first INSERT statement) is inserted into the table T1.

Table 159. Sequence of statements to demonstrate effect of ROLLBACK TO SAVEPOINT statement

Statement	Savepoint ID
INSERT INTO T1 VALUES (3)	1
SAVEPOINT A	2
INSERT INTO T1 VALUES (5)	2
SAVEPOINT B	3
INSERT INTO T1 VALUES (6)	3
ROLLBACK TO SAVEPOINT A	2
COMMIT	

WITH DATA option

Not all input values are audited when you specify the WITH DATA option. LOB, LONG, XML and structured type parameters appear as NULL.

Date, time, and timestamp fields are recorded in ISO format.

If WITH DATA is specified in one policy, but WITHOUT DATA is specified in another policy associated with objects involved in the execution of the SQL statement, then WITH DATA takes precedence and data is audited for that particular statement. For example, if the audit policy associated with a user specifies WITHOUT DATA, but the policy associated with a table specifies WITH DATA, when that user accesses that table, the input data used for the statement is audited.

You are not able to determine which rows were modified on a positioned-update or positioned-delete statement. Only the execution of the underlying SELECT statement is logged, not the individual FETCH. It is not possible from the EXECUTE record to determine which row the cursor is on when the statement is issued. When replaying the statement at a later time, it is only possible to issue the SELECT statement to see what range of rows may have been affected.

Example of replaying past activities

Consider in this example that as part of their comprehensive security policy, a company requires that they retain the ability to retroactively go back up to seven years to analyze the effects of any particular request against certain tables in their database. To do this, they institute a policy of archiving their weekly backups and associated log files such that they can reconstitute the database for any chosen moment in time. They require that the database audit capture sufficient information about every request made against the database to allow the replay and analysis of any request against the relevant, restored database. This requirement covers both static and dynamic SQL statements.

This example shows the audit policy that must be in place at the time the SQL statement is issued, and the steps to archive the audit logs and later to extract and analyze them.

1. Create an audit policy that audits the EXECUTE category and apply this policy to the database:

   ```
   CREATE AUDIT POLICY STATEMENTS CATEGORIES EXECUTE WITH DATA
      STATUS BOTH ERROR TYPE AUDIT
   COMMIT

   AUDIT DATABASE USING POLICY STATEMENTS
   COMMIT
   ```

2. Regularly archive the audit log to create an archive copy.

   The following statement should be run by the security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT_ARCHIVE stored procedure, on a regular basis, for example, once a week or once a day, depending on the amount of data logged. These archived files can be kept for whatever period is required. The AUDIT_ARCHIVE procedure is called with two input parameters: the path to the archive directory and -2, to indicate that the archive should be run on all members:

   ```
   CALL SYSPROC.AUDIT_ARCHIVE( '/auditarchive', -2 )
   ```

3. The security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT_LIST_LOGS table function, uses AUDIT_LIST_LOGS to examine all of the available audit logs from April 2006, to determine which logs may contain the necessary data:

   ```
   SELECT FILE FROM TABLE(SYSPROC.AUDIT_LIST_LOGS('/auditarchive'))
      AS T WHERE FILE LIKE 'db2audit.dbname.log.0.200604%'
   FILE
   --------------------------------------
   ...
   db2audit.dbname.log.0.20060418235612
   db2audit.dbname.log.0.20060419234937
   db2audit.dbname.log.0.20060420235128
   ```

4. From this output, the security administrator observes that the necessary logs should be in one file: `db2audit.dbname.log.20060419234937`. The timestamp shows this file was archived at the end of the day for the day the auditors want to see.

The security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure, uses this filename as input to AUDIT_DELIM_EXTRACT to extract the audit data into delimited files. The audit data in these files can be loaded into DB2 database tables, where it can be analyzed to find the particular statement the auditors are interested in. Even though the auditors are only interested in a single SQL statement, multiple statements from the unit of work may need to be examined in case they have any impact on the statement of interest.

5. In order to replay the statement, the security administrator must take the following actions:
   - Determine the exact statement to be issued from the audit record.
   - Determine the user who issued the statement from the audit record.
   - Re-create the exact permissions of the user at the time they issued the statement, including any LBAC protection.
   - Reproduce the compilation environment, by using the compilation environment column in the audit record in combination with the SET COMPILATION ENVIRONMENT statement.
   - Restore the database to its exact state at the time the statement was issued.

   To avoid disturbing the production system, any restore of the database and replay of the statement should be done on a second database system. The security administrator, running as the user who issued the statement, can reissue the statement as found in the statement text with any input variables that are provided in the statement value data elements.

Part 8. Appendixes

Appendix A. Overview of the DB2 technical information

DB2 technical information is available in multiple formats that can be accessed in multiple ways.

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command-line help
  - Command help
  - Message help

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hardcopy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this email address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss. English and translated DB2 Version 10.1 manuals in PDF format can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg27009474.

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

Note: The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

Table 160. DB2 technical information

Name	Form Number	Available in print	Last updated
Administrative API Reference	SC27-3864-00	Yes	April, 2012
Administrative Routines and Views	SC27-3865-01	No	January, 2013
Call Level Interface Guide and Reference Volume 1	SC27-3866-01	Yes	January, 2013
Call Level Interface Guide and Reference Volume 2	SC27-3867-01	Yes	January, 2013
Command Reference	SC27-3868-01	Yes	January, 2013
Database Administration Concepts and Configuration Reference	SC27-3871-01	Yes	January, 2013
Data Movement Utilities Guide and Reference	SC27-3869-01	Yes	January, 2013
Database Monitoring Guide and Reference	SC27-3887-01	Yes	January, 2013
Data Recovery and High Availability Guide and Reference	SC27-3870-01	Yes	January, 2013
Database Security Guide	SC27-3872-01	Yes	January, 2013
DB2 Workload Management Guide and Reference	SC27-3891-01	Yes	January, 2013
Developing ADO.NET and OLE DB Applications	SC27-3873-01	Yes	January, 2013
Developing Embedded SQL Applications	SC27-3874-01	Yes	January, 2013
Developing Java Applications	SC27-3875-01	Yes	January, 2013
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-3876-00	No	April, 2012
Developing RDF Applications for IBM Data Servers	SC27-4462-00	Yes	January, 2013
Developing User-defined Routines (SQL and External)	SC27-3877-01	Yes	January, 2013
Getting Started with Database Application Development	GI13-2046-01	Yes	January, 2013

Table 160. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
Getting Started with DB2 Installation and Administration on Linux and Windows	GI13-2047-00	Yes	April, 2012
Globalization Guide	SC27-3878-00	Yes	April, 2012
Installing DB2 Servers	GC27-3884-01	Yes	January, 2013
Installing IBM Data Server Clients	GC27-3883-00	No	April, 2012
Message Reference Volume 1	SC27-3879-01	No	January, 2013
Message Reference Volume 2	SC27-3880-01	No	January, 2013
Net Search Extender Administration and User's Guide	SC27-3895-01	No	January, 2013
Partitioning and Clustering Guide	SC27-3882-01	Yes	January, 2013
Preparation Guide for DB2 10.1 Fundamentals Exam 610	SC27-4540-00	No	January, 2013
Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611	SC27-4541-01	No	January, 2013
pureXML Guide	SC27-3892-01	Yes	January, 2013
Spatial Extender User's Guide and Reference	SC27-3894-00	No	April, 2012
SQL Procedural Languages: Application Enablement and Support	SC27-3896-01	Yes	January, 2013
SQL Reference Volume 1	SC27-3885-01	Yes	January, 2013
SQL Reference Volume 2	SC27-3886-01	Yes	January, 2013
Text Search Guide	SC27-3888-01	Yes	January, 2013
Troubleshooting and Tuning Database Performance	SC27-3889-01	Yes	January, 2013
Upgrading to DB2 Version 10.1	SC27-3881-01	Yes	January, 2013
What's New for DB2 Version 10.1	SC27-3890-01	Yes	January, 2013
XQuery Reference	SC27-3893-01	No	January, 2013

Table 161. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
DB2 Connect Installing and Configuring DB2 Connect Personal Edition	SC27-3861-00	Yes	April, 2012

Table 161. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print	Last updated
DB2 Connect Installing and Configuring DB2 Connect Servers	SC27-3862-01	Yes	January, 2013
DB2 Connect User's Guide	SC27-3863-01	Yes	January, 2013

Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure

To start SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.
For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Accessing different versions of the DB2 Information Center

Documentation for other versions of DB2 products is found in separate information centers on ibm.com®.

About this task

For DB2 Version 10.1 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v10r1.

For DB2 Version 9.8 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/.

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/.

For DB2 Version 9.5 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

For DB2 Version 9.1 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

For DB2 Version 8 topics, go to the *DB2 Information Center* URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

Updating the DB2 Information Center installed on your computer or intranet server

A locally installed DB2 Information Center must be updated periodically.

Before you begin

A DB2 Version 10.1 Information Center must already be installed. For details, see the "Installing the DB2 Information Center using the DB2 Setup wizard" topic in *Installing DB2 Servers*. All prerequisites and restrictions that applied to installing the Information Center also apply to updating the Information Center.

About this task

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates update existing Information Center features and languages. One benefit of automatic updates is that the Information Center is unavailable for a shorter time compared to during a manual update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.
- Manual updates can be used to update existing Information Center features and languages. Automatic updates reduce the downtime during the update process, however you must use the manual process when you want to add features or languages. For example, a local Information Center was originally installed with both English and French languages, and now you want to also install the German language; a manual update will install German, as well as, update the existing Information Center features and languages. However, a manual update requires you to manually stop, update, and restart the Information Center. The Information Center is unavailable during the entire update process. In the automatic update process the Information Center incurs an outage to restart the Information Center after the update only.

This topic details the process for automatic updates. For manual update instructions, see the "Manually updating the DB2 Information Center installed on your computer or intranet server" topic.

Procedure

To automatically update the DB2 Information Center installed on your computer or intranet server:

1. On Linux operating systems,
   a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V10.1` directory.
   b. Navigate from the installation directory to the `doc/bin` directory.
   c. Run the `update-ic` script:
      ```
      update-ic
      ```
2. On Windows operating systems,
   a. Open a command window.
   b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `<Program Files>\IBM\DB2 Information Center\Version 10.1` directory, where `<Program Files>` represents the location of the Program Files directory.
   c. Navigate from the installation directory to the `doc\bin` directory.
   d. Run the `update-ic.bat` file:
      ```
      update-ic.bat
      ```

Results

The DB2 Information Center restarts automatically. If updates were available, the Information Center displays the new and updated topics. If Information Center updates were not available, a message is added to the log. The log file is located in `doc\eclipse\configuration` directory. The log file name is a randomly generated number. For example, `1239053440785.log`.

Manually updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

About this task

Updating your locally installed *DB2 Information Center* manually requires that you:

1. Stop the *DB2 Information Center* on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to apply updates. The Workstation version of the DB2 Information Center always runs in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you must install, you can use the Update feature to obtain and install them

   Note: If your environment requires installing the *DB2 Information Center* updates on a machine that is not connected to the internet, mirror the update site to a local file system by using a machine that is connected to the internet and has the *DB2 Information Center* installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.
   If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.
3. Stop the stand-alone Information Center, and restart the *DB2 Information Center* on your computer.

Note: On Windows 2008, Windows Vista (and higher), the commands listed later in this section must be run as an administrator. To open a command prompt or graphical tool with full administrator privileges, right-click the shortcut and then select **Run as administrator**.

Procedure

To update the *DB2 Information Center* installed on your computer or intranet server:

1. Stop the *DB2 Information Center*.
   - On Windows, click **Start** > **Control Panel** > **Administrative Tools** > **Services**. Then right-click **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv10 stop
     ```
2. Start the Information Center in stand-alone mode.
   - On Windows:
     a. Open a command window.

    b. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the `Program_Files\IBM\DB2 Information Center\Version 10.1` directory, where *Program_Files* represents the location of the `Program Files` directory.
    c. Navigate from the installation directory to the `doc\bin` directory.
    d. Run the `help_start.bat` file:

       ```
       help_start.bat
       ```

  - On Linux:
    a. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the `/opt/ibm/db2ic/V10.1` directory.
    b. Navigate from the installation directory to the `doc/bin` directory.
    c. Run the `help_start` script:

       ```
       help_start
       ```

   The systems default Web browser opens to display the stand-alone Information Center.

3. Click the **Update** button ( ). (JavaScript must be enabled in your browser.) On the right panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check that the selections you want to install, then click **Install Updates**.
5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
   - On Windows, navigate to the `doc\bin` directory within the installation directory, and run the `help_end.bat` file:

     ```
     help_end.bat
     ```

     Note: The `help_end` batch file contains the commands required to safely stop the processes that were started with the `help_start` batch file. Do not use `Ctrl-C` or any other method to stop `help_start.bat`.
   - On Linux, navigate to the `doc/bin` directory within the installation directory, and run the `help_end` script:

     ```
     help_end
     ```

     Note: The `help_end` script contains the commands required to safely stop the processes that were started with the `help_start` script. Do not use any other method to stop the `help_start` script.
7. Restart the *DB2 Information Center*.
   - On Windows, click **Start** > **Control Panel** > **Administrative Tools** > **Services**. Then right-click **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv10 start
     ```

Results

The updated *DB2 Information Center* displays the new and updated topics.

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 database products. Lessons provide step-by-step instructions.

Before you begin

You can view the XHTML version of the tutorial from the Information Center at http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials

To view the tutorial, click the title.

"pureXML" in *pureXML Guide*
Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 database products.

DB2 documentation
Troubleshooting information can be found in the *Troubleshooting and Tuning Database Performance* or the Database fundamentals section of the *DB2 Information Center*, which contains:

- Information about how to isolate and identify problems with DB2 diagnostic tools and utilities.
- Solutions to some of the most common problem.
- Advice to help solve other problems you might encounter with your DB2 database products.

IBM Support Portal
See the IBM Support Portal if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the IBM Support Portal at http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows

Terms and conditions

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

COPYRIGHT LICENSE:

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

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _*enter the year or years*_. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Celeron, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

Index

Printed in USA

SC27-4541-01

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611