IBM DB2 10.1
for Linux, UNIX, and Windows

Database Administration Concepts and Configuration Reference

Updated January, 2013

SC27-3871-01

IBM DB2 10.1
for Linux, UNIX, and Windows

Database Administration Concepts and Configuration Reference

Updated January, 2013

SC27-3871-01

Note
Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 891.

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).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

About this book

The *Database Administration Concepts and Configuration Reference* provides information about database planning and design, and implementation and management of database objects. This book also contains reference information for database configuration and tuning.

Who should use this book

This book is intended primarily for database and system administrators who need to design, implement and maintain a database to be accessed by local or remote clients. It can also be used by programmers and other users who require an understanding of the administration and operation of the DB2® relational database management system.

How this book is structured

This book is structured in four parts. Parts 1 through 3 provide a conceptual overview of the DB2 database product, starting with general concepts about data servers, and working progressively toward explanations of the objects that commonly comprise DB2 databases. Part 4 contains reference information.

Part 1. Data Servers
This section briefly describes DB2 data servers, including management of their capacity and large page support in 64-bit environments on AIX®. In addition, it also provides information on running multiple DB2 copies on a single computer, information on the automatic features that assist you in managing your database system, information on designing, creating, and working with instances, and optional information on configuring Lightweight Directory Access Protocol (LDAP) servers.

Part 2. Databases
This section describes the design, creation, and maintenance of databases, buffer pools, table spaces, and schemas. Detailed information about database partitions is found in the *Partitioning and Clustering Guide*.

Part 3. Database objects
This section describes the design, creation, and maintenance of the following database objects: tables, constraints, indexes, triggers, sequences and views.

Part 4. Reference
This section contains reference information for configuring and tuning your database system with environment and registry variables, and configuration parameters. It also lists the various naming rules and SQL and XML limits.

Part 1. Data servers

Chapter 1. DB2 data servers

Data servers provide software services for the secure and efficient management of structured information. DB2 is a hybrid relational and XML data server.

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/.

Management of data server capacity

If data server capacity does not meet your present or future needs, you can expand its capacity by adding disk space and creating additional containers, or by adding memory. If these simple strategies do not add the capacity you need, also consider adding processors or physical partitions.

When you scale your system by changing the environment, be aware of the impact that such a change can have on your database procedures such as loading data, or backing up and restoring databases.

Adding processors

If a single-partition database configuration with a single processor is used to its maximum capacity, you might either add processors or add logical partitions. The advantage of adding processors is greater processing power. In a single-partition database configuration with multiple processors (SMP), processors share memory and storage system resources. All of the processors are in one system, so communication between systems and coordination of tasks between systems does not factor into the workload. Utilities such as load, back up, and restore can take advantage of the additional processors.

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

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

- Default degree (**dft_degree**)
- Maximum degree of parallelism (**max_querydegree**)
- Enable intrapartition parallelism (**intra_parallel**)

You should also evaluate parameters that determine how applications perform parallel processing.

In an environment where TCP/IP is used for communication, review the value for the **DB2TCPCONNMGRS** registry variable.

Adding additional computers

If you have an existing partitioned database environment, you can increase processing power and data-storage capacity by adding additional computers (either single-processor or multiple-processor) and storage resource to the environment. The memory and storage resources are not shared among computers. This choice provides the advantage of balancing data and user access across storage and computers.

After adding the new computers and storage, you would use the **START DATABASE MANAGER** command to add new database partition servers to the new computers. A new database partition is created and configured for each database in the instance on each new database partition server that you add. In most situations, you do not need to restart the instance after adding the new database partition servers.

Enabling large page support (AIX)

To enable large page support in DB2 database systems on AIX operating systems, you must configure some operating system parameters and then set the **DB2_LARGE_PAGE_MEM** registry variable.

Before you begin

You must have root authority to work with the AIX operating system commands.

About this task

In addition to the traditional page size of 4 KB, the POWER4 processors (and higher) on System z® also support a 16 MB page size. Applications such as IBM DB2 Version 10.1 for AIX, that require intensive memory access and that use large amounts of virtual memory can gain performance improvements by using large pages.

Note:

1. Setting the **DB2_LARGE_PAGE_MEM** registry variable also implies that the memory is pinned.
2. You should be extremely cautious when configuring your system for pinning memory and supporting large pages. Pinning too much memory results in heavy paging activities for the memory pages that are not pinned. Allocating too much physical memory to large pages degrades system performance if there is insufficient memory to support the 4 KB pages.

Restrictions

Enabling large pages prevents the self-tuning memory manager from automatically tuning overall database memory consumption, so it should only be considered for well-defined workloads that have relatively static database memory requirements.

Procedure

To enable large page support in DB2 database systems on AIX operating systems:

1. Configure your AIX server for large page support by issuing the **vmo** command with the following flags:

   ```
   vmo -r -o lgpg_size=LargePageSize -o lgpg_regions=LargePages
   ```

   Where *LargePageSize* specifies the size in bytes of the hardware-supported large pages, and *LargePages* specifies the number of large pages to reserve. For example, if you want to allocate 25 GB for large page support, run the command as follows:

   ```
   vmo -r -o lgpg_size=16777216 -o lgpg_regions=1600
   ```

   For detailed instructions on how to run the **vmo** command, refer to your AIX manuals.
2. Run the **bosboot** command so that the **vmo** command that you previously ran takes effect following the next system boot.
3. After the server comes up, enable it for pinned memory. Issue the **vmo** command with the following flags:

   ```
   vmo -o v_pinshm=1
   ```
4. Use the **db2set** command to set the **DB2_LARGE_PAGE_MEM** registry variable to DB, then start the DB2 database manager. For example:

   ```
   db2set DB2_LARGE_PAGE_MEM=DB
   db2start
   ```

Results

When these steps are complete, the DB2 database system directs the operating system to use large page memory for the database shared memory region.

Pinning DB2 database shared memory (AIX)

To pin DB2 database shared memory on AIX operating systems, you must configure some operating system parameters and then set the **DB2_PINNED_BP** registry variable.

Before you begin

You must have root authority to perform the AIX operating system commands.

About this task

The advantage of having portions of memory pinned is that when you access a page that is pinned, you can retrieve the page without going through the page replacement algorithm. A disadvantage is that you must take care to ensure that the system is not overcommitted, as the operating system will have reduced flexibility in managing memory. Pinning too much memory results in heavy paging activities for the memory pages that are not pinned.

Restrictions

If you set the **DB2_PINNED_BP** registry variable to YES, self tuning for database shared memory cannot be enabled.

Procedure

To pin DB2 database shared memory on AIX operating systems:

1. Configure the AIX operating system to enable pinned memory. Issue the **vmo** command with the following flags:

   ```
   vmo -o v_pinshm=1
   ```

   For detailed instructions on how to run the **vmo** command, refer to your AIX manuals.
2. (Optional) If you are using medium sized pages (which is the default behavior), ensure that the DB2 instance owner has the CAP_BYPASS_RAC_VMM and CAP_PROPAGATE capabilities. For example:

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

   where *db2inst1* is the DB2 instance owner user ID.
3. Run the **bosboot** command so that the **vmo** command will take effect following the next system boot.
4. After the server comes up, enable the DB2 database system for pinned memory.
   a. Issue the **db2set** command to set the **DB2_PINNED_BP** registry variable to YES.
   b. Start the DB2 database manager.

   For example:

   ```
   db2set DB2_PINNED_BP=YES
   db2start
   ```

Results

When these steps are complete, the DB2 database system directs the operating system to pin the DB2 database shared memory.

Chapter 2. Multiple DB2 copies overview

With Version 9 and later, you can install and run multiple DB2 copies on the same computer. A DB2 copy refers to one or more installations of DB2 database products in a particular location on the same computer. Each DB2 Version 9 copy can be at the same or different code levels.

The benefits of doing this include:

- The ability to run applications that require different DB2 database versions on the same computer at the same time
- The ability to run independent copies of DB2 database products for different functions
- The ability to test on the same computer before moving the production database to the latter version of the DB2 database product
- For independent software vendors, the ability to embed a DB2 database server product into your product and hide the DB2 database from your users. For COM+ applications, use and distribute the IBM Data Server Driver for ODBC and CLI with your application instead of the Data Server Runtime Client as only one Data Server Runtime Client can be used for COM+ applications at a time. The IBM Data Server Driver for ODBC and CLI does not have this restriction.

Table 1 lists the relevant topics in each category.

Table 1. Overview to multiple DB2 copies information

Category	Related topics
General information and restrictions	• "Default IBM database client interface copy"
Installation	• "Installing DB2 servers (Linux and UNIX)" in *Installing DB2 Servers* • "Installing DB2 servers (Windows)" in *Installing DB2 Servers*
Configuration	• "Setting the DAS when running multiple DB2 copies" on page 10 • "Setting the default instance when using multiple DB2 copies (Windows)" on page 12 • "dasupdt - Update DAS command" in *Command Reference*
Administration	• "Updating DB2 copies (Windows)" on page 15 • "Updating DB2 copies (Linux and UNIX)" on page 14 • "db2iupdt - Update instances command" in *Command Reference* • "db2swtch - Switch default DB2 copy command" in *Command Reference* • "db2SelectDB2Copy API - Select the DB2 copy to be used by your application" in *Administrative API Reference*

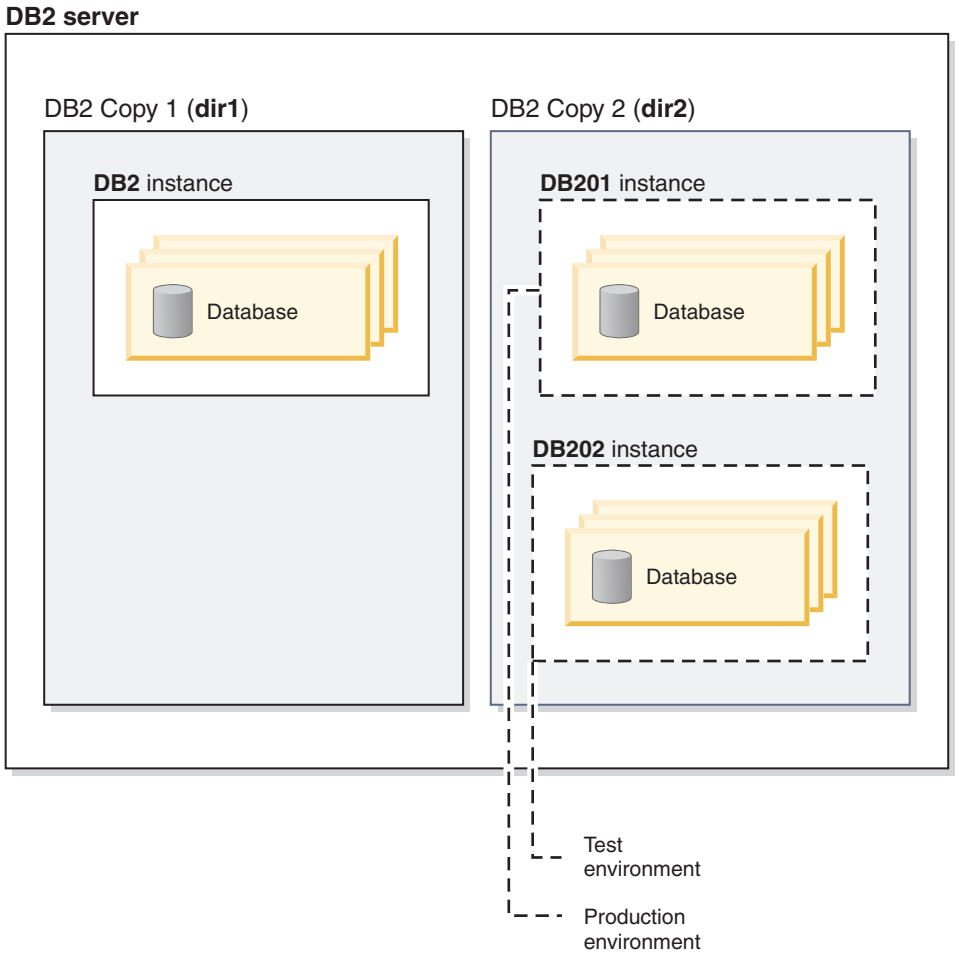
Default IBM database client interface copy

You can have multiple DB2 copies on a single computer, as well as a default IBM database client interface copy, which is the means by which a client application has the ODBC driver, CLI driver, and .NET data provider code needed to interface with the database by default.

In Version 9.1 (and later), the code for the IBM database client interface copy is included with the DB2 copy. With Version 9.5 (and later) there is a new product you can choose to install that has the needed code to allow a client application to interface with a database. This product is IBM Data Server Driver Package (DSDRIVER). With Version 9.5 (and later) you can install DSDRIVER on an IBM data server driver copy separate from the installation of a DB2 copy.

Following Version 9.1, you can have multiple DB2 copies installed on your computer; following Version 9.5, you can have multiple IBM database client interface copies and multiple DB2 copies installed on your computer. During the time of installation of a new DB2 copy or new IBM data server driver copy you would have had the opportunity to change the default DB2 copy and the default IBM database client interface copy.

The following diagram shows multiple DB2 copies installed on a DB2 server, which can be any combination of the DB2 database products:



Version 8 and Version 9 (or later) copies can coexist on the same computer, however Version 8 must be the default DB2 and IBM database client interface copy. You cannot change from the Version 8 copy to the Version 9 (or later) copy as the default DB2 copy or default IBM database client interface copy during installation, nor can you later run the switch default copy command, **db2swtch**, unless you first upgrade to Version 9 (or later) or uninstall Version 8 copy. If you run the **db2swtch**

command when Version 8 exists on the system, you will receive an error message indicating that you cannot change the default copy because Version 8 is found on the system.

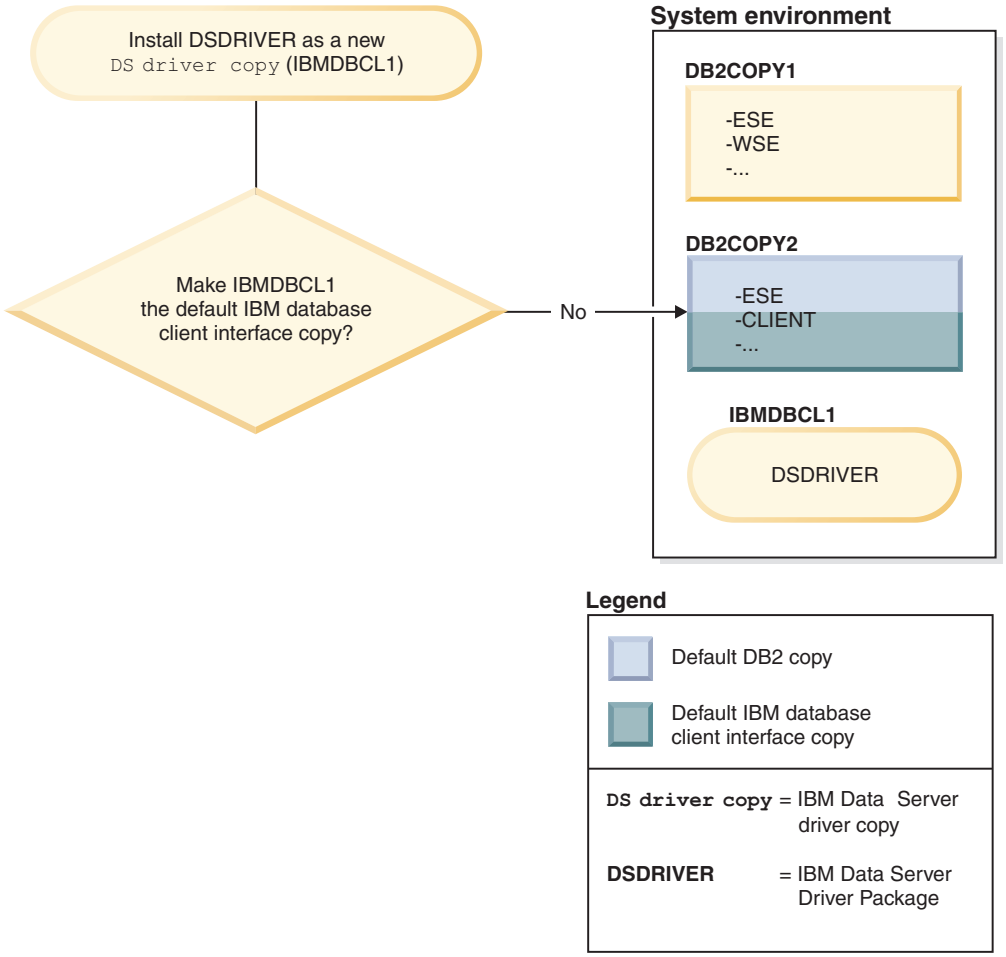
Sometime after installing multiple DB2 copies or multiple IBM data server driver copies, you might want to change either the default DB2 copy or the default IBM database client interface copy. If you have Version 8 installed, you must uninstall the product or upgrade it to Version 9 (or later) before you can change the default DB2 copy, or change the default IBM database client interface copy.

Client applications can always choose to go directly to a data server driver location which is the directory where the DSDRIVER is installed.

When you uninstall either the DB2 copy or the IBM data server driver copy that had been the default IBM database client interface copy, the defaults are managed for you. Chosen default copies are removed and new defaults are selected for you. When you uninstall the default DB2 copy which is not the last DB2 copy on the system, you will be asked to switch the default to another DB2 copy first.

Choosing a default when installing a new IBM database client interface copy

Following Version 9.5, consider the scenario where you have installed two DB2 copies (DB2COPY1 and DB2COPY2). DB2COPY2 is the default DB2 copy and the default IBM database client interface copy.

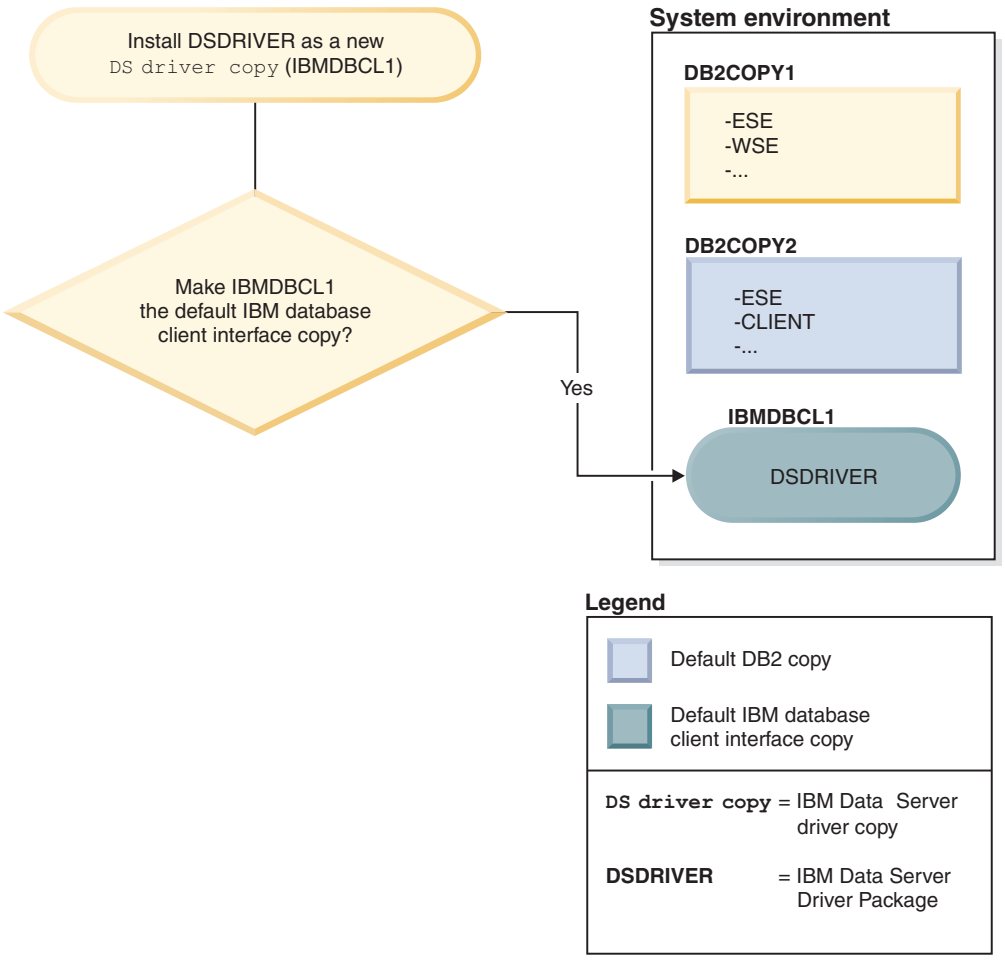


Install IBM Data Server Driver Package (DSDRIVER) on a new IBM data server driver copy.

During the install of the new IBM data server driver copy (IBMDBCL1) you are asked if you want to make the new IBM data server driver copy the default IBM database client interface copy.

If you respond "No", then DB2COPY2 remains the default IBM database client interface copy. (And it continues to be the default DB2 copy.)

However, consider the same scenario but you respond "Yes" when asked if you want to make the new IBM data server driver copy the default IBM database client interface copy.



In this case, IBMDBCL1 becomes the default IBM database client interface copy. (DB2COPY2 remains the default DB2 copy.)

Setting the DAS when running multiple DB2 copies

Starting with Version 9.1, you can have multiple DB2 copies running on the same computer. This affects how the DB2 Administration Server (DAS) operates.

About this task

The DAS is a unique component within the database manager that is limited to having only one version active, despite how many DB2 copies are installed on the same computer. For this reason, the following restrictions and functional requirements apply.

Restrictions

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 .

On the server, there can be only one DAS version and it administers instances as follows:

- If the DAS runs on Version 9.1 or Version 9.5, then it can administer Version 8, Version 9.1, or Version 9.5 instances.
- If the DAS runs on Version 8, then it can administer only Version 8 instances. You can upgrade your Version 8 DAS, or drop it and create a new Version 9.5 DAS to administer the Version 8 or later instances.

Only one DAS can be created on a given computer at any given time despite the number of DB2 copies that are installed on the same computer. This DAS will be used by all the DB2 copies that are on the same computer. In Version 9.1 or later, the DAS can belong to any DB2 copy that is currently installed.

If the DAS is running in a Version 9.5 copy and you want it to run in another Version 9.5 copy, use the **dasupdt** command. If the DAS is running in a Version 8, Version 9.1, or Version 9.5 copy and you want it to run in a Version 9.7 copy, you cannot use **dasupdt**, use the **dasmigr** command to upgrade the DAS from to Version 9.7.

On Windows operating systems, you can also use the **dasupdt** command when you need to run the DAS in a new Default DB2 copy of the same version.

Procedure

To set up the DAS in one of the DB2 copies:

Choose one of the following actions:

- If the DAS is not created, then create a DAS in one of the DB2 copies.
- Use the **dasupdt** command only to update the DAS so that it runs in another DB2 copy of the same release.
- Use the **dasmigr** command to upgrade from Version 8, Version 9.1, or Version 9.5 to Version 9.7 DAS.

Setting the default instance when using multiple DB2 copies (Windows)

The **DB2INSTANCE** environment is set according to the DB2 copy that your environment is currently set up to use. If you do not set it explicitly to an instance in the current copy, it defaults to the default instance that is specified with the **DB2INSTDEF** profile registry variable.

About this task

DB2INSTDEF is the default instance variable that is specific to the current DB2 copy in use. Every DB2 copy has its own **DB2INSTDEF** profile registry variable. Instance names must be unique on the system; when an instance is created, the database manager scans through existing copies to ensure its uniqueness.

Use the following guidelines to set the default instance when using multiple DB2 copies:

- If **DB2INSTANCE** is not set for a particular DB2 copy, then the value of **DB2INSTDEF** is used for that DB2 copy. This means:
  - If **DB2INSTANCE**=ABC and **DB2INSTDEF**=XYZ, ABC is the value that is used
  - If **DB2INSTANCE** *is not set* and **DB2INSTDEF**=XYZ, XYZ is used
  - If **DB2INSTANCE** *is not set* and **DB2INSTDEF** *is not set*, then any application or command that depends on a valid **DB2INSTANCE** will not work.
- You can use either the **db2envar.bat** command or the db2SelectDB2Copy API to switch DB2 copies. Setting all the environment variables appropriately (for example, **PATH**, **INCLUDE**, **LIB**, and **DB2INSTANCE**) will also work, but you must ensure that they are set properly.

  Note: Using the **db2envar.bat** command is not quite the same as setting the environment variables. The **db2envar.bat** command determines which DB2 copy it belongs to, and then adds the path of this DB2 copy to the front of the **PATH** environment variable.

  When there are multiple DB2 copies on the same computer, the **PATH** environment variable can point to only one of them: the DEFAULT COPY. For example, if DB2COPY1 is under `c:\sqllib\bin` and is the default copy; and DB2COPY2 is under `d:\sqllib\bin`. If you want to use DB2COPY2 in a regular command window, you would run `d:\sqllib\bin\db2envar.bat` in that command window. This adjusts the **PATH** (and some other environment variables) for this command window so that it picks up binaries from `d:\sqllib\bin`.
- **DB2INSTANCE** is only valid for instances under the DB2 copy that you are using. However, if you switch copies by running the **db2envar.bat** command, **DB2INSTANCE** is updated to the value of **DB2INSTDEF** for the DB2 copy that you switched to initially.
- **DB2INSTANCE** is the current DB2 instance that is used by applications that are executing in that DB2 copy. When you switch between copies, by default, **DB2INSTANCE** is changed to the value of **DB2INSTDEF** for that copy. **DB2INSTDEF** is less meaningful on a one copy system because all the instances are in the current copy; however, it is still applicable as being the default instance, if another instance is not set.
- All global profile registry variables are specific to a DB2 copy, unless you specify them using `SET VARIABLE=`*`variable_name`*.

Multiple instances of the database manager

Multiple instances of the database manager might be created on a single server. This means that you can create several instances of the same product on a physical computer, and have them running concurrently. This provides flexibility in setting up environments.

Note: The same instance name cannot be used in two different DB2 copies.

You might want to have multiple instances to create the following environments:

- Separate your development environment from your production environment.
- Separately tune each environment for the specific applications it will service.
- Protect sensitive information from administrators. For example, you might want to have your payroll database protected on its own instance so that owners of other instances are not able to see payroll data.

Note:

- On UNIX operating systems only: To prevent environmental conflicts between two or more instances, ensure that each instance home directory is on a local file system.
- On Windows operating systems only: Instances are cataloged as either local or remote in the node directory. Your default instance is defined by the **DB2INSTANCE** environment variable. You can **ATTACH** to other instances to perform maintenance and utility tasks that can be done only at an instance level, such as creating a database, forcing off applications, monitoring a database, or updating the database manager configuration. When you attempt to attach to an instance that is not in your default instance, the node directory is used to determine how to communicate with that instance.
- On any platform: DB2 database program files are physically stored at one location and each instance points back to the copy to which that instance belongs so that the program files are not duplicated for each instance that is created. Several related databases can be located within a single instance.

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.

Updating DB2 copies (Linux and UNIX)

You can update an existing DB2 copy and all instances running on that copy. You can also choose to install a new DB2 copy and selectively update instances to run on this new copy after installation.

Before you begin

- Ensure that you have root user authority.
- Download and uncompress a fix pack. The fix pack and the DB2 copy that you want to update must be of the same release.

About this task

Follow these instructions to update your DB2 copies from one fix pack level to another (within the same version level) or to install additional functionality.

If you have DB2 Version 8, Version 9.1, Version 9.5 or Version 9.7 copies, you cannot update these copies from previous releases to DB2 Version 9.8, you need to upgrade them.

Restrictions

- You will not be able to update more than one DB2 copy at the same time. In order to update other DB2 copies that might be installed on the same computer, you must rerun the installation.

Procedure

To update your DB2 copies:

1. Log on with root user authority.
2. Stop all DB2 processes.

3. Update each DB2 copy using one of the following choices:
   - To update an existing DB2 copy and update all the instances running on this DB2 copy, issue the **installFixPack** command. You cannot install additional functionality with this command.
   - To install a new DB2 copy and selectively update the instances running on an existing DB2 copy to the new copy after installation, issue the **db2setup** command and select **Install New** in the **Install a Product** panel. To install a new copy, you can also perform a response file installation specifying a new location as installation path. Any of these options allow you to also install additional functionality.
   - To add functionality to an existing DB2 copy, select **Work with Existing** in the **Install a Product** panel. Then select the DB2 copy that you want to update with the **Add new function** action. This action is only available when the DB2 copy is at the same release level as the install image. To add functionality, you can also perform a response file installation or issue the **db2_install** command.

   Important: The command **db2_install** is deprecated and might be removed in a future release. Use the **db2setup** command with a response file instead.
4. If you installed a new DB2 copy, use the **db2iupdt** command to update any instances that are running in a different DB2 copy of the same release that you want them to run under the new copy. The following table shows several examples of updating instances:

Instance	DB2 copy	Example to update to another copy
db2inst1	/opt/IBM/db2/V9.1/	cd /opt/IBM/db2/V9.1_FP3/instance ./db2iupdt db2inst1
db2inst2	/opt/IBM/db2/V9.5FP2/	cd /home/db2/myV9.5_FP1/instance ./db2iupdt -D db2inst2[a]
db2inst3	/opt/IBM/db2/V9.7/	cd /home/db2/myV9.7/instance ./db2iupdt -k db2inst3[b]

Note:

a. Use the **-D** parameter to update an instance from a later release level copy to a earlier release level copy.

b. Use the **-k** parameter to keep the current instance type during the update to a DB2 copy that has a higher level of instance type. If you updated from WSE to ESE, when you update the instance without this parameter the instance type wse is converted to ese.

Results

Once you have installed or updated a DB2 copy, you can always update instances that run in other DB2 copies of the same release, to run on this new DB2 copy by issuing the **db2iupdt** command.

Updating DB2 copies (Windows)

You can update an existing DB2 copy and all instances running on that copy to a new fix pack level. You can also choose to install a new DB2 copy and selectively update instances to run on this new copy after installation.

Before you begin

- Ensure that you have Local Administrator authority.
- Download and uncompress a fix pack. The fix pack and the DB2 copy that you want to update must be of the same release.

About this task

Follow these instructions to update your DB2 copies from one fix pack level to another (within the same version level) or to install additional functionality.

Restrictions

- You can only update an instance of the same release from a earlier release level copy to a later release level copy. You cannot update an instance from a later release level copy to a earlier release level copy.
- You will not be able to update more than one DB2 copy at the same time. In order to update other DB2 copies that might be installed on the same computer, you must rerun the installation.
- Coexistence of a 32-bit DB2 data server and a 64-bit DB2 data server on the same Windows x64 computer is not supported. It is not possible to upgrade directly from a 32-bit x64 DB2 installation at Version 8 to a 64-bit installation at Version 9.8.

Procedure

To update your DB2 copies:

1. Log on as a user with Local Administrator authority.
2. Stop all DB2 instances, services and applications.
3. Run `setup.exe` to launch the DB2 wizard to install a DB2 copy. You have the following choices:
   - To update an existing DB2 copy and update all the instances running on this DB2 copy, select **Work with Existing** in the **Install a Product** panel. Then select the DB2 copy that you want to update with the **update** action. You cannot install additional functionality with this action.
   - To install a new DB2 copy and selectively update the instances running on an existing DB2 copy to the new copy after installation, select **Install New** in the **Install a Product** panel. This option allows you to also install additional functionality.
   - To add functionality to an existing DB2 copy, select **Work with Existing** in the **Install a Product** panel. Then select the DB2 copy that you want to update with the **Add new function** action. This action is only available when the DB2 copy is at the same release level as the install image.
4. If you installed a new DB2 copy, use the **db2iupdt** command to update any instances that are running in a different DB2 copy of the same release that you want them to run under the new copy. The following table shows several examples of updating instances:

Instance	DB2 copy	Example to update to another copy
db2inst1	`C:\Program Files\IBM\SQLLIB_91\BIN`	`cd D:\Program Files\IBM\SQLLIB_91_FP5\BIN` `db2iupdt db2inst1 /u: user-name,password`
db2inst2	`C:\Program Files\IBM\SQLLIB_97\BIN`	`cd D:\Program Files\IBM\SQLLIB_97\BIN` `db2iupdt db2inst2 /u: user-name,password`

Results

Once you have installed or updated a DB2 copy, you can always update instances that run in other DB2 copies of the same release, to run on this new DB2 copy by issuing the **db2iupdt** command.

Running multiple instances concurrently (Windows)

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

Procedure

1. To run multiple instances concurrently in the same DB2 copy, using the command line:
   a. Set the **DB2INSTANCE** variable to the name of the other instance that you want to start by entering:
      ```
      set db2instance=another_instName
      ```
   b. Start the instance by entering the **db2start** command.
2. To run multiple instances concurrently in different DB2 copies, use either of the following methods:
   - Using the DB2 Command window from the **Start** > **Programs** > **IBM DB2** > *DB2 Copy Name* > **Command Line Tools** > **DB2 Command Window**: the Command window is already set up with the correct environment variables for the particular DB2 copy chosen.
   - Using **db2envar.bat** from a Command window:
     a. Open a Command window.
     b. 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
        ```

   After you switch to a particular DB2 copy, use the method specified in the preceding section, "To run multiple instances concurrently in the same DB2 copy", to start the instances.

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`

Chapter 3. Autonomic computing overview

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.

"Automatic features" on page 21 provides a high-level summary of the capabilities that comprise the DB2 autonomic computing environment; the following table provides a more detailed, categorized overview of the product's autonomic capabilities:

Table 2. Overview of autonomic computing information

Category	Related topics
Self-tuning memory	• "Memory usage" in *Troubleshooting and Tuning Database Performance* • "Self-tuning memory" in *Troubleshooting and Tuning Database Performance* • "Self-tuning memory overview" in *Troubleshooting and Tuning Database Performance* • "auto_maint - Automatic maintenance" on page 746 • "db_storage_path - Automatic storage path monitor element" in *Database Monitoring Guide and Reference* • "num_db_storage_paths - Number of automatic storage paths monitor element" in *Database Monitoring Guide and Reference* • "tablespace_using_auto_storage - Using automatic storage monitor element" in *Database Monitoring Guide and Reference* • "Configuring memory and memory heaps" on page 38 • "Agent, process model, and memory configuration overview" on page 41 • "Shared file handle table" on page 45 • "Running vendor library functions in fenced-mode processes" on page 45 • "ADMIN_GET_MEM_USAGE table function - Get total memory consumption for instance" in *Administrative Routines and Views* • "Agent and process model configuration" on page 41 • "Configuring databases across multiple partitions" on page 43
Automatic storage	• "Databases use automatic storage by default" on page 46 • "Automatic storage table spaces" on page 161 • "Automatic re-sizing of DMS table spaces" on page 157
Data compression	• "Data compression" on page 46 – "Table compression" on page 294 – "Index compression" on page 430 – "Backup compression" in *Data Recovery and High Availability Guide and Reference* • "Table-level compression dictionary creation" on page 304 • "Compression dictionary creation during load operations" in *Data Movement Utilities Guide and Reference*

Table 2. Overview of autonomic computing information (continued)

Category	Related topics
Automatic database backup	• "Automatic database backup" in *Data Recovery and High Availability Guide and Reference* • "Enabling automatic backup" in *Data Recovery and High Availability Guide and Reference* • "Developing a backup and recovery strategy" in *Data Recovery and High Availability Guide and Reference*
Automatic reorganization	"Automatic reorganization" in *Troubleshooting and Tuning Database Performance*
Automatic statistics collection	• "Automatic statistics collection" in *Troubleshooting and Tuning Database Performance* • "Using automatic statistics collection" in *Troubleshooting and Tuning Database Performance* • "Storage used by automatic statistics collection and profiling" in *Troubleshooting and Tuning Database Performance* • "Automatic statistics collection activity logging" in *Troubleshooting and Tuning Database Performance*
Configuration Advisor	• "Generating database configuration recommendations" on page 52 – "Tuning configuration parameters using the Configuration Advisor" on page 52 – "Example: Requesting configuration recommendations using the Configuration Advisor" on page 53 – "AUTOCONFIGURE command" in *Command Reference* – "AUTOCONFIGURE command using the ADMIN_CMD procedure" in *Administrative Routines and Views* – "db2AutoConfig API - Access the Configuration Advisor" in *Administrative API Reference* • "Quick-start tips for performance tuning" in *Troubleshooting and Tuning Database Performance*
Health monitor	• "Health monitor" in *Database Monitoring Guide and Reference* • "Health indicator process cycle" in *Database Monitoring Guide and Reference* – "Enabling health alert notification" in *Database Monitoring Guide and Reference* – "Configuring health indicators using a client application" in *Database Monitoring Guide and Reference* • "Health indicators summary" in *Database Monitoring Guide and Reference*

Table 2. Overview of autonomic computing information (continued)

Category	Related topics
Utility throttling	• "Utility throttling" on page 55 • "Asynchronous index cleanup" in *Troubleshooting and Tuning Database Performance* • "Asynchronous index cleanup for MDC tables" in *Troubleshooting and Tuning Database Performance* – "LIST UTILITIES command" in *Command Reference* – "SET UTIL_IMPACT_PRIORITY command" in *Command Reference* – "util_impact_lim - Instance impact policy" on page 737 – "utility_priority - Utility Priority monitor element" in *Database Monitoring Guide and Reference*

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.

Health monitor

The health monitor is a server-side tool that proactively monitors situations or changes in your database environment that could result in a performance degradation or a potential outage. A range of health information is presented without any form of active monitoring on your part. If a health risk is encountered, the database manager informs you and advises you on how to proceed. The health monitor gathers information about the system by using the snapshot monitor and does not impose a performance penalty. Further, it does not turn on any snapshot monitor switches to gather information.

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.

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.

Self-tuning memory overview

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

enabled, the memory tuner dynamically distributes available memory resources among the following memory consumers: buffer pools, locking memory, package cache, and sort memory.

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

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 1 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.

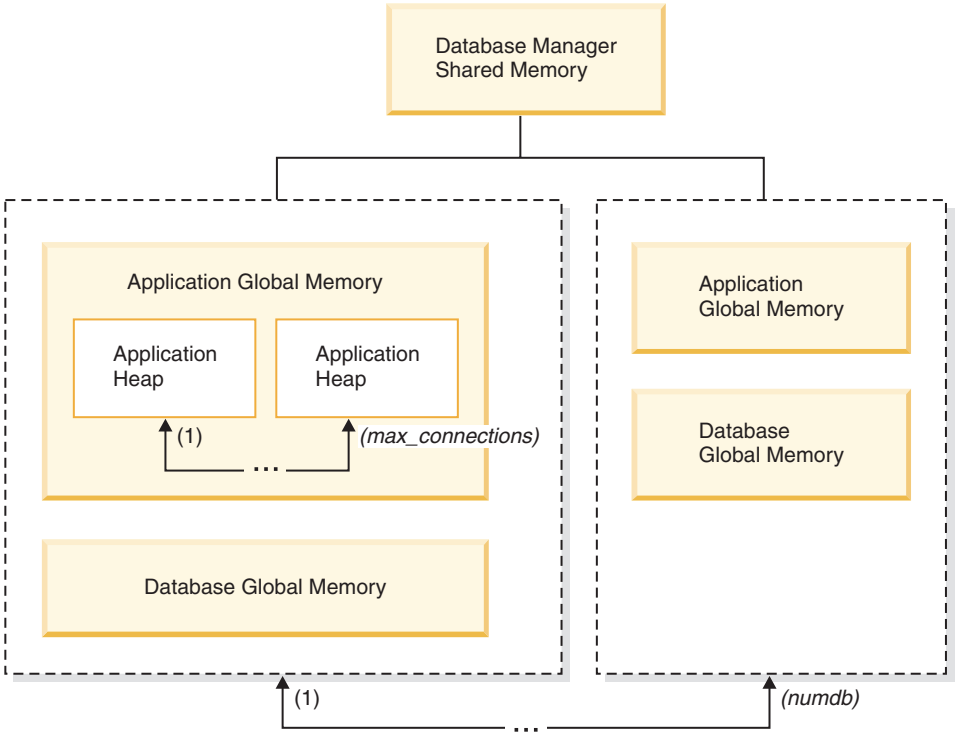


Figure 1. Types of memory allocated by the database manager

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**.

Memory parameter interaction and limitations

Although you can enable self-tuning memory and use the default AUTOMATIC setting for most memory-related configuration parameters, it might be useful to know the limitations of the different memory parameters and the interactions between them, in order to have more control over their settings and to understand why out-of-memory errors are still possible under certain conditions.

Memory types

Basically, the DB2 database manager uses two types of memory:

Performance memory
This is memory used to improve database performance. Performance memory is controlled and distributed to the various performance heaps by the self-tuning memory manager (STMM). You can set the **database_memory** configuration parameter to a maximum amount of performance memory or set **database_memory** to AUTOMATIC to let STMM manage the overall amount of performance memory.

Functional memory
This is used by application programs. You can use the **appl_memory** configuration parameter to control the maximum amount of functional

memory, or application memory, that is allocated by DB2 database agents to service application requests. By default, this parameter is set to AUTOMATIC, meaning that functional memory requests are allowed as long as there are system resources available. If you are using DB2 database products with memory usage restrictions or if you set **instance_memory** to a specific value, an **instance_memory** limit is enforced and functional memory requests are allowed if the total amount of memory allocated by the database partition is within the **instance_memory** limit.

Before the AUTOMATIC setting was available, various operating system and DB2 tools were available that allowed you to see the amount of space used by different types memory, such as shared memory, private memory, buffer pool memory, locklists, sort memory (heaps), and so forth, but it was almost impossible to see the total amount of memory used by the DB2 database manager. If one of the heaps reached the memory limit, a statement in an application would fail with an out-of-memory error message. If you increased the memory for that heap and reran the application, you might then have received an out-of-memory error on another statement for another heap. Now, you can remove hard upper limits on individual functional memory heaps by using the default AUTOMATIC configuration parameter setting.

If required (to avoid scenarios where a poorly behaving database application requires extremely large amounts of memory), you can apply a limit on overall application memory at the database level by using the **appl_memory** configuration parameter. You can also apply a limit for an individual heap by changing the appropriate database configuration parameter for that heap from the AUTOMATIC setting to a fixed value. If all of the configuration parameters for all of the functional memory heaps are set to AUTOMATIC and an **instance_memory** limit is enforced, the only limit on application memory consumption is the **instance_memory** limit . If you also set **instance_memory** to AUTOMATIC and you are using a DB2 database product with memory usage restrictions, the DB2 database manager automatically determines an upper limit on memory consumption.

You can easily see the total amount of instance memory consumed and the current **instance_memory** consumption by using the **db2pd -dbptnmem**command or the ADMIN_GET_MEM_USAGE table function.

Interactions between memory configuration parameters

When self-tuning memory manager (STMM) is active and self-tuning of database memory is enabled (**database_memory** is set to AUTOMATIC), STMM checks the free memory available on the system and automatically determines how much memory to dedicate to performance heaps for optimal performance. All the performance heaps contribute to the overall **database_memory** size. In addition to the performance memory requirements, some memory is required to ensure the operation and integrity of the DB2 database manager. The difference between the space used by **instance_memory** and the space required by these two memory consumers is available for application memory (**appl_memory**) use. Functional memory for application programs is then allocated as needed. If an **instance_memory** limit is not enforced, there are no additional restrictions on how much memory a single application can allocate.

Depending on the configuration, STMM also periodically queries how much free system memory is remaining and how much free **instance_memory** space is remaining if there is an **instance_memory** limit. To prevent application failures, STMM prioritizes application requirements ahead of performance criteria. If

required, it degrades performance by decreasing the amount of space available for performance heaps, thus providing enough free system memory and **instance_memory** space to meet application memory requests. As applications are completed, the used memory is freed, ready to be reused by other applications or to be reclaimed for **database_memory** use by STMM. If performance of the database system becomes unacceptable during periods of heavy application activity, it might be useful either to apply controls on how many applications the database manager is allowed to run (by using either the connection concentrator or the new Workload Management feature of DB2 Version 9.5) or to add memory resources to the system.

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
```

Self-tuning memory in a DB2 pureScale environment

When self-tuning memory is enabled in a DB2 pureScale environment, the tuning member monitors the memory configuration and propagates any configuration changes to all other members.

When self-tuning memory is enabled in a DB2 pureScale environment, there is a single member (known as the tuning member) that monitors the memory configuration and propagates any configuration changes to all other members to maintain a consistent configuration across all the members in the instance.

In a DB2 pureScale environment, when the tuning member is specified as -1, the tuning member is randomly selected every time the database is activated. Also, if the member on which the tuner is running deactivates, the tuner will automatically start on another member that is currently able to run the tuner. In order for a member to run the tuner, the database must be active on that member and that member must have **self_tuning_mem** set to ON.

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

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

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

  ```
  CALL SYSPROC.ADMIN_CMD('update stmm tuning member membernum')
  ```

  The tuning member can be switched between having the tuning member randomly selected and using an explicitly specified tuning member as required.

Note that when the tuning member changes, some data collected from the member which was running the tuner, is discarded. This data must be recollected on the new tuning member. During this short period of time when the data is being recollected, the memory tuner will still tune the system; however, the tuning can occur slightly differently than it did on the original member.

Starting the memory tuner in a DB2 pureScale environment

In a DB2 pureScale environment, the memory tuner will run whenever the database is active on one or more members that have **self_tuning_mem** set to ON.

Disabling self-tuning memory for a specific member

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

An ALTER BUFFERPOOL statement that specifies the size of a buffer pool on a particular member will create an exception entry (or update an existing entry) for that buffer pool in the SYSCAT.BUFFERPOOLEXCEPTIONS 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 used 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 member 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

The workload that is run on each member is expected to have similar memory requirements. The memory requirements can be skewed across members, for example, if resource-intensive sorts are only performed on one member or, if some members are associated with different hardware and more available memory than others. Self tuning memory can still be activated on some members. To take advantage of self-tuning memory in heterogeneous memory environments, identify a set of members that have similar memory requirements and activate the self tuning memory on those members. The self-tuning memory can then be disabled on the remaining members and their memory can be configured manually.

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.
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.

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

Configuration parameter name	Description	Meaning of the AUTOMATIC setting
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.
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		

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

Configuration parameters	Set to **AUTOMATIC** upon instance upgrade or creation	Set to **AUTOMATIC** upon database upgrade	Set to **AUTOMATIC** upon database creation
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.

Agent, process model, and memory configuration overview

DB2 data servers exploit multithreaded architecture on both 32-bit and 64-bit platforms to provide you with a number of benefits, such as enhanced usability, better sharing of resources, memory footprint reduction, and consistent threading architecture across all operating systems.

The following table lists the agent, process, and memory configuration topics by category:

Table 5. Overview of agent, process, and memory configuration information

Category	Related topics
General information, restrictions, and incompatibilities	• "Configuring memory and memory heaps" on page 38 • "Agent and process model configuration" • "The DB2 Process Model" in *Troubleshooting and Tuning Database Performance* • "Configuring databases across multiple partitions" on page 43
Installation and upgrade	• "Connection concentrator" in *DB2 Connect User's Guide* • "DB2 Connect™ tuning" in *DB2 Connect User's Guide* • "Considerations for OS/390® and zSeries® SYSPLEX exploitation" in *DB2 Connect User's Guide* • "Disk and memory requirements" in *Installing DB2 Servers* • "Modifying kernel parameters (Linux)" in *Installing DB2 Servers* • "DB2 server behavior changes" in *Upgrading to DB2 Version 10.1*

Table 5. Overview of agent, process, and memory configuration information (continued)

Category	Related topics
Performance	• "Connection-concentrator improvements for client connections" in *Troubleshooting and Tuning Database Performance* • "Database agents" in *Troubleshooting and Tuning Database Performance* • "Database agent management" in *Troubleshooting and Tuning Database Performance* • "Database manager shared memory" in *Troubleshooting and Tuning Database Performance* • "Memory allocation in DB2" in *Troubleshooting and Tuning Database Performance* • "Tuning memory allocation parameters" in *Troubleshooting and Tuning Database Performance*
Commands, APIs, registry variables, functions, and routines	• "db2pd - Monitor and troubleshoot DB2 database command" in *Command Reference* • "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference* • "RESET DATABASE MANAGER CONFIGURATION command " in *Command Reference* • "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference* • "db2mtrk - Memory tracker command" in *Command Reference* • "sqlfupd data structure" in *Administrative API Reference* • • "Shared file handle table" on page 45 • "Running vendor library functions in fenced-mode processes" on page 45 • "ADMIN_GET_MEM_USAGE table function - Get total memory consumption for instance" in *Administrative Routines and Views* • "SQL and XML limits" in *SQL Reference Volume 1* • "SYSCAT.PACKAGES catalog view" in *SQL Reference Volume 1* • "DBMCFG administrative view - Retrieve database manager configuration parameter information" in *Administrative Routines and Views* • "ADMIN_CMD procedure-Run administrative commands" in *Administrative Routines and Views*

Table 5. Overview of agent, process, and memory configuration information (continued)

Category	Related topics
Configuration parameters	• "Configuration parameters summary" on page 635 • "appl_memory - Application Memory configuration parameter" on page 744 • "applheapsz - Application heap size" on page 745 • "database_memory - Database shared memory size" on page 762 • "dbheap - Database heap" on page 764 • "instance_memory - Instance memory" on page 699 • "locklist - Maximum storage for lock list" on page 790 • "max_connections - Maximum number of client connections" on page 705 • "max_coordagents - Maximum number of coordinating agents" on page 706 • "maxappls - Maximum number of active applications" on page 806 • "mon_heap_sz - Database system monitor heap size" on page 711 • "num_poolagents - Agent pool size" on page 714 • "stat_heap_sz - Statistics heap size" on page 845 • "stmtheap - Statement heap size" on page 847
Monitor elements	• "Agents and connections" in *Database Monitoring Guide and Reference* • "agents_from_pool - Agents Assigned From Pool" in *Database Monitoring Guide and Reference* • "agents_registered - Agents Registered" in *Database Monitoring Guide and Reference* • "agents_registered_top - Maximum Number of Agents Registered" in *Database Monitoring Guide and Reference* • "agents_stolen - Stolen Agents" in *Database Monitoring Guide and Reference* • "appls_in_db2 - Applications Executing in the Database Currently" in *Database Monitoring Guide and Reference* • "associated_agents_top - Maximum Number of Associated Agents" in *Database Monitoring Guide and Reference* • "coord_agents_top - Maximum Number of Coordinating Agents" in *Database Monitoring Guide and Reference* • "local_cons - Local Connections" in *Database Monitoring Guide and Reference* • "local_cons_in_exec - Local Connections Executing in the Database Manager" in *Database Monitoring Guide and Reference* • "num_gw_conn_switches - Maximum Agent Overflows" in *Database Monitoring Guide and Reference* • "rem_cons_in - Remote Connections To Database Manager" in *Database Monitoring Guide and Reference* • "rem_cons_in_exec - Remote Connections Executing in the Database Manager" in *Database Monitoring Guide and Reference*

Configuring databases across multiple partitions

The database manager provides a single view of all database configuration elements across multiple partitions. This means that you can update or reset a

database configuration across all database partitions without invoking the **db2_all** command against each database partition.

You can update a database configuration across partitions by issuing only one SQL statement or only one administration command from any partition on which the database resides. By default, the method of updating or resetting a database configuration is *on all database partitions*.

For backward compatibility of command scripts and applications, you have three options:

- Use the **db2set** command to set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable to TRUE, as follows:

  ```
  DB2_UPDDBCFG_SINGLE_DBPARTITION=TRUE
  ```

  Note: Setting the registry variable does not apply to **UPDATE DATABASE CONFIGURATION** or **RESET DATABASE CONFIGURATION** requests that you make using the ADMIN_CMD procedure.
- Use the **DBPARTITIONNUM** parameter with either the **UPDATE DATABASE CONFIGURATION** or the **RESET DATABASE CONFIGURATION** command or with the ADMIN_CMD procedure. For example, to update the database configurations on all database partitions, call the ADMIN_CMD procedure as follows:

  ```
  CALL SYSPROC.ADMIN_CMD
    ('UPDATE DB CFG USING sortheap 1000')
  ```

  To update a single database partition, call the ADMIN_CMD procedure as follows:

  ```
  CALL SYSPROC.ADMIN_CMD
    ('UPDATE DB CFG DBPARTITIONNUM 10 USING sortheap 1000')
  ```
- Use the **DBPARTITIONNUM** parameter with the db2CfgSet API. The flags in the **db2Cfg** structure indicate whether the value for the database configuration is to be applied to a single database partition. If you set a flag, you must also provide the **DBPARTITIONNUM** value, for example:

  ```
  #define db2CfgSingleDbpartition          256
  ```

  If you do not set the db2CfgSingleDbpartition value, the value for the database configuration applies to all database partitions unless you set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable to TRUE or you set *versionNumber* to anything that is less than the version number for Version 9.5, for the db2CfgSet API that sets the database manager or database configuration parameters.

When upgrading your databases to Version 9.7, existing database configuration parameters, as a general rule, retain their values after database upgrade. However, new parameters are added using their default values and some existing parameters are set to their new Version 9.7 default values. Refer to the "DB2 server behavior changes" topic in *Upgrading to DB2 Version 10.1* for details about the changes to existing database configuration parameters. Any subsequent update or reset database configuration requests for the upgraded databases will apply to all database partitions by default.

For existing update or reset command scripts, the same rules mentioned previously apply to all database partitions. You can modify your scripts to include the **DBPARTITIONNUM** option of the **UPDATE DATABASE CONFIGURATION** or **RESET DATABASE CONFIGURATION** command, or you can set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable.

For existing applications that call the **db2CfgSet** API, you must use the instructions for Version 9.5 or later. If you want the pre-Version 9.5 behavior, you can set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable or modify your applications to call the API with the Version 9.5 or later version number, including the new `db2CfgSingleDbpartition` flag and the new **dbpartitionnum** field to update or reset database configurations for a specific database partition.

Note: If you find that database configuration values are inconsistent, you can update or reset each database partition individually.

Shared file handle table

The threaded database manager maintains a single shared file handle table for each database and all agents working on each database so that I/O requests made on the same file do not require the file to be reopened and closed.

Before Version 9.5, the file handle table was maintained separately by each DB2 agent, and the size of the per-agent file handle table was controlled by the **maxfilop** configuration parameter. Starting in Version 9.5, the database manager maintains a single shared file handle table for the entire database, such that the same file handle can be shared among all agents working on the same database file. As a result, the **maxfilop** configuration parameter is used to control the size of the shared file handle table.

Because of this change, the **maxfilop** configuration parameter has a different default value and new minimum and maximum values starting in Version 9.5. During database upgrade, the **maxfilop** configuration parameter is automatically set to this default value if you are upgrading from a release before Version 9.5.

Running vendor library functions in fenced-mode processes

The database manager supports vendor library functions in fenced-mode processes that perform such tasks as data compression, TSM backups, and log data archiving.

About this task

Prior to Version 9.5, vendor library functions, vendor utilities, or routines were run inside agent processes. Since Version 9.5, because the DB2 database manager itself is a multithreaded application, vendor library functions that are no longer threadsafe and cause memory or stack corruption or, worse, data corruption in DB2 databases. For these reasons, a new fenced-mode process is created for each invocation of a vendor utility, and vendor library functions or routines run inside this fenced-mode process. This does not result in significant performance degradation.

Note: The fenced-mode feature is not available for Windows platforms.

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 165 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]

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

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.

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.

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 6. 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.

Starting with DB2 Version 9, you can use the Configuration Advisor to determine the initial configuration for new databases. The Configuration Advisor recommends the **auto_stmt_stats** database configuration parameter be set to ON.

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.

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.

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.

Generating database configuration recommendations

The Configuration Advisor is automatically run when you create a database. You can also run the Configuration Advisor by specifying the **AUTOCONFIGURE** command in the command line processor (CLP) or by calling the db2AutoConfig API.

Procedure

To request configuration recommendations by using the CLP, enter the following command:

```
AUTOCONFIGURE
     USING input_keyword param_value
     APPLY value
```

Example

Following is an example of an **AUTOCONFIGURE** command that requests configuration recommendations based on input about how the database is used but specifies that the recommendations not be applied:

```
    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
```

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 2

```
       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 2. 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. 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 3 shows the hierarchical relationship among systems, instances, and databases.

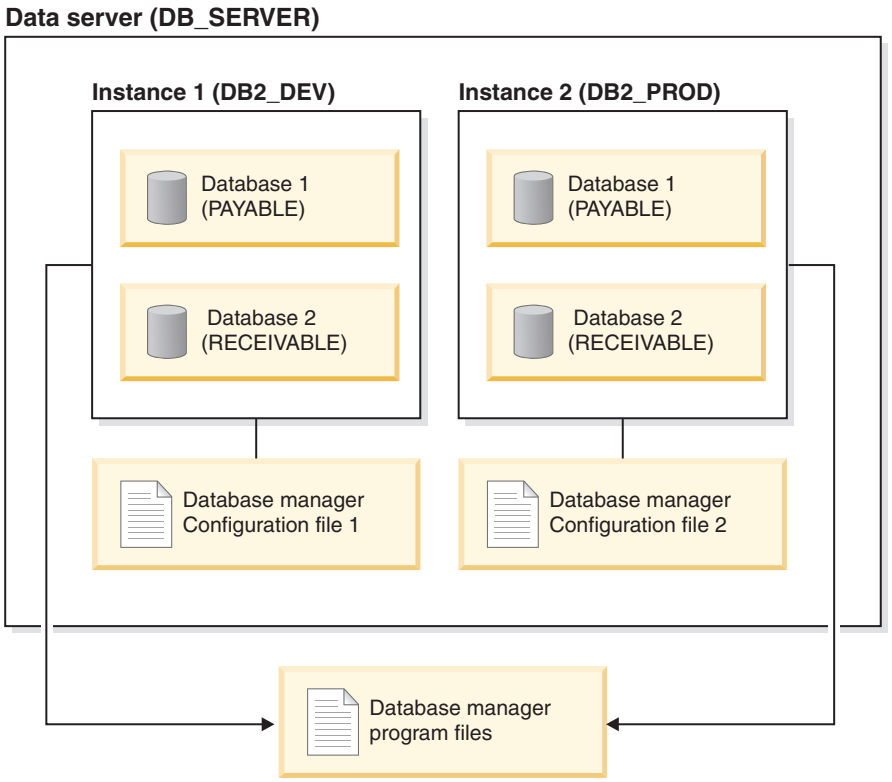


Figure 3. Hierarchical relationship among DB2 systems, instances, and databases

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.

Managing instances

When working with instances, you can start or stop instances, and attach to or detach from instances.

About this task

Each instance is managed by users who belong to the **sysadm_group** defined in the *instance configuration file*, also known as the *database manager configuration file*. Creating user IDs and user groups is different for each operating environment.

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, enter 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.

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 546.

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.

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 546.

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 **db2start** 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.

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.

Instance management in a DB2 pureScale environment

This section contains information about administering a DB2 pureScale instance. The topics covered here are specific to the DB2 pureScale Feature and do not cover the administration of the broader DB2 database product.

This section provides information about essential administration concepts, tasks, and user scenarios for these areas:

- Starting and stopping members and cluster caching facilities.
- Maintenance tasks such as upgrading cluster caching facilities or member hosts, and adding resources to the shared file system cluster.
- Automated restart of failed members and cluster caching facilities
- Configuring the shared file system cluster, cluster caching facilities, and buffer pools.

Multiple active databases in a DB2 pureScale environment

As of DB2 Version 9.8 Fix Pack 3, you can now have multiple active databases in a DB2 pureScale environment.

In previous iterations of DB2 Version 9.8, the DB2 pureScale environment differed from the DB2 Enterprise Server Edition and partitioned database environments in that only one database could be active at any one time. With the release of DB2 Version 9.8 Fix Pack 3, that restriction has been lifted.

The user experience in a DB2 pureScale environment is now virtually identical to the experience with multiple active databases in DB2 Enterprise Server Edition and partitioned database environments.

Outside of a DB2 pureScale environment, the maximum number of databases that can be active at any given time is 255. However, in a DB2 pureScale environment, the maximum number of databases that can be active at any given time is 200.

The default maximum number of active databases in a DB2 pureScale environment is 32. This is also the new default for DB2 Enterprise Server Edition and partitioned database environments.

To have multiple active databases in a DB2 pureScale environment, see the configuration parameter changes outlined in "DB2 pureScale CF memory parameter configuration" on page 868

To change the number of active databases in a DB2 pureScale environment, you modify the **numdb** configuration parameter, and the change comes into effect after the next global restart.

In addition to the **numdb** limit in a DB2 pureScale environment, there is a maximum number of database activations across all members in an instance. This maximum number is 512 database activations. As an example, if each member in a four member DB2 pureScale environment activated 200 databases, that is a total of 800 database member activations. Since 800 database activations exceeds the maximum upper limit, an error is returned.

In a multiple database environment, if member crash recovery (MCR) is required, the number of databases that will be recovered in parallel on each member is set by the value of the **numdb** configuration parameter or the **DB2_MCR_RECOVERY_PARALLELISM_CAP** registry variable, whichever value is smaller.

Starting and stopping cluster components and databases in a DB2 pureScale environment

In a DB2 pureScale environment, a cluster caching facility or a member is started or stopped as a part of a global **db2start** or **db2stop** command.

When you issue the **db2start** or **db2stop** command, all of the currently defined members and cluster caching facilities will be started or stopped. However, in some situations, it might be useful to start and stop these cluster components and databases at a more granular level.

Starting a cluster caching facility

A cluster caching facility (CF) is started as part of a global **db2start** or an individual **db2start** CF command. This task focuses on starting a single cluster caching facility.

About this task

On a global **db2start**, the cluster manager attempts to start the primary role (also known as the PRIMARY state) on the preferred primary cluster caching facility, and the other cluster caching facility will be started in the secondary role (also known as PEER state).

If one cluster caching facility is already started and running in the PRIMARY state, the next cluster caching facility that you start (the secondary cluster caching facility) will enter a CATCHUP phase which ensures that the secondary cluster caching facility has a copy of all pertinent information from the primary cluster caching facility in its (the secondary cluster caching facility's) memory before it transitions into PEER state.

Procedure

To start a specific cluster caching facility:

Issue the command:

```
db2start CF CF-identifier
```

Example

John, a DBA, has added a second cluster caching facility to a DB2 pureScale instance. He queries the status of all the cluster caching facilities in the instance at this point, using

```
SELECT ID,
       varchar(CURRENT_HOST,10) AS CUR_HOST,
       varchar(STATE,17) AS STATE,
       ALERT
FROM SYSIBMADM.DB2_CF
```

and gets the output:

```
ID     CUR_HOST     STATE      ALERT
------ ------------ ---------- -----
   128 so5          PRIMARY    NO
   129 so6          STOPPED    NO

     2 record(s) selected.
```

The cluster caching facility 128 is the only active cluster caching facility, so it is also in the PRIMARY state.

John issues `db2start CF 129` and queries the status of the cluster caching facilities, using

```
SELECT ID,
       varchar(CURRENT_HOST,10) AS CUR_HOST,
       varchar(STATE,17) AS STATE,
       ALERT
FROM SYSIBMADM.DB2_CF
```

and gets the output:

```
ID     CUR_HOST     STATE        ALERT
------ ------------ ----------   -----
   128 so5          PRIMARY      NO
   129 so6          CATCHUP(50%) NO

     2 record(s) selected.
```

The cluster caching facility 129 is now getting a copy of all pertinent information from cluster caching facility 128 so that it can take over as the primary cluster caching facility if cluster caching facility 128 fails.

He queries the status of all the cluster caching facilities, using

```
SELECT ID,
       varchar(CURRENT_HOST,10) AS CUR_HOST,
       varchar(STATE,17) AS STATE,
       ALERT
FROM SYSIBMADM.DB2_CF
```

and gets the output:

```
ID     CUR_HOST     STATE      ALERT
------ ------------ ---------- -----
   128 so5          PRIMARY    NO
   129 so6          PEER       NO

     2 record(s) selected.
```

Now that cluster caching facility 129 is in PEER state, it is available to take over as the primary cluster caching facility if the current primary fails.

Stopping a cluster caching facility

A cluster caching facility (CF) is stopped as part of a global **db2stop** or an individual **db2stop CF** command. This topic focuses on stopping a single cluster caching facility.

About this task

Issue the **db2stop CF** command if you want to stop a cluster caching facility on a host while keeping other instance processes on the same host up and running. You can use **db2stop CF** before shutting down a host.

Restrictions

You cannot stop the primary cluster caching facility if any of these situations are true:

- There are active members in the instance.
- The primary cluster caching facility contains dirty pages.
- The primary cluster caching facility contains locks.
- The secondary cluster caching facility is not in PEER state.

You can, however, stop the secondary cluster caching facility using the **FORCE** option, even if any of these situations are true.

Procedure

To stop a specific cluster caching facility, issue this command:

```
db2stop CF CF-identifier
```

Starting a member

DB2 members are started as part of a global **db2start** or an individual **db2start** command. This topic focuses on starting a single member.

About this task

When you issue a **db2start** (either for the instance or member) the database manager starts all the DB2 idle processes on the host and all of the cluster caching facilities (with active hosts) defined in the instance, if the idle processes and cluster caching facilities are not already running. If the cluster caching facilities cannot be started, the member start operation will fail. If the idle processes cannot be started on the member's home host, the start operation will fail but DB2 cluster services will start the member in restart light mode on another host. (See Restart light for more information.)

Procedure

To start a specific member, issue this command:

```
db2start member member-id
```

Results

The database manager starts individual members on the host that is currently specified in the db2nodes.cfg file, even if that host is not the home host for the target member. In other words, a member that was previously running as a guest member on another host will be started on that host in restart light mode. If that member's home host is active and ready to receive its resident member, DB2 cluster services will fail the member back to its home host.

Example

John, a DBA, has finished performing maintenance on a host with one member (member 0) defined on it. He has now restarted the instance on the host (`db2start instance on host so1`). The member was shut down before the maintenance operations. He then queries the status of all members in the instance using

```
SELECT ID,
       varchar(HOME_HOST,10) AS HOME_HOST,
       varchar(CURRENT_HOST,10) AS CUR_HOST,
       varchar(STATE,21) AS STATE,
       ALERT
FROM SYSIBMADM.DB2_MEMBER
```

and gets the output:

```
ID     HOME_HOST CUR_HOST     STATE      ALERT
------ --------- ------------ ---------- -----
     0 so1       so1          STOPPED    NO
     2 so2       so2          STARTED    NO
     4 so3       so3          STARTED    NO

     3 record(s) selected.
```

John issues this command to start member 0: `db2start member 0`. He queries the status of all members in the instance using

```
SELECT ID,
       varchar(HOME_HOST,10) AS HOME_HOST,
       varchar(CURRENT_HOST,10) AS CUR_HOST,
       varchar(STATE,21) AS STATE,
       ALERT
FROM SYSIBMADM.DB2_MEMBER
```

and gets the output:

```
ID     HOME_HOST CUR_HOST     STATE      ALERT
------ --------- ------------ ---------- -----
     0 so1       so1          STARTED    NO
     2 so2       so2          STARTED    NO
     4 so3       so3          STARTED    NO

     3 record(s) selected.
```

Stopping a member

DB2 members are stopped as part of a global **db2stop** or an individual **db2stop** command. This topic focuses on stopping a single member.

About this task

Issue the `db2stop member` command if you want to stop a member on a host but keep other instance processes on the host up and running. This means that other members or cluster caching facilities on the same host will not be affected. However, note that stopping a member is not a sufficient prerequisite for performing maintenance on the host, because the host still remains a viable failover target for other members.

Restrictions

You cannot stop a member if there are any active database connections on the member. If the member is in restart light mode and has not yet completed member crash recovery (that is, at least one of the databases is inconsistent on that member), it cannot be stopped using a **db2stop** command. However, it can be stopped by using the **FORCE** option. It is strongly recommended that you do not stop a member undergoing crash recovery.

Procedure

To stop a member, issue this command:

```
db2stop member member-id
```

Database activation with the DB2 pureScale Feature

A database in a DB2 pureScale environment can be activated explicitly with the **ACTIVATE DATABASE** command, or implicitly when the first client connects.

Before you begin

Ensure that you have one of these authorization levels:

- SYSMAINT
- SYSCTRL
- SYSADM

About this task

In a DB2 pureScale environment, after the last user disconnects the database remains activated. In order to shut the database down, you must issue a **DEACTIVATE DATABASE** command or, **db2stop** command.

Procedure

To activate a database in a DB2 pureScale environment, issue the **ACTIVATE DATABASE** command which will automatically activate the database across all members.

Example

To activate the TEST database across all members, run the command:

```
DB2 ACTIVATE DATABASE TEST
```

Database deactivation with the DB2 pureScale Feature

In a DB2 pureScale environment, a database that has been explicitly activated through the **ACTIVATE DATABASE** command or, implicitly activated by a user connection, can only be deactivated through a **DEACTIVATE DATABASE** command.

Before you begin

Ensure that you have one of these authorization levels:

- SYSMAINT
- SYSCTRL
- SYSADM

About this task

In a DB2 pureScale environment, after the last user disconnects the database remains activated. In order to shut the database down, you must issue a **DEACTIVATE DATABASE** command or, **db2stop** command.

Procedure

1. To deactivate a database in a DB2 pureScale environment, issue the **DEACTIVATE DATABASE** command which will automatically deactivate the database across all members.
2. To deactivate a specific member issue the **MEMBER** parameter of the **DEACTIVATE DATABASE** command.

Example

Example 1

To deactivate the TEST database, run this command:

```
DB2 DEACTIVATE DATABASE TEST
```

Example 2

To deactivate a specific member, run this command:

```
DB2 DEACTIVATE DATABASE TEST MEMBER 10
```

Maintenance in a DB2 pureScale environment

One of the advantages of the IBM DB2 pureScale Feature is that it maintains continuous availability of the database during planned maintenance activities.

DB2 cluster services allows you to perform rolling maintenance on the cluster caching facilities in your DB2 pureScale instance. Similarly, members or member hosts that are targeted for maintenance can be easily removed from and reintegrated to the DB2 pureScale instance.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for putting or removing target hosts from maintenance mode. 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.

Performing maintenance on a member host

You can perform maintenance on or apply updates to a member host without affecting the availability of the databases in the DB2 pureScale instance.

Before you begin

Any DB2 server processes and any other processes accessing the file systems managed by DB2 cluster services must be shut down prior to running this task. All instances must be stopped on the host before the host can enter maintenance mode. When the instance is stopped on the host, the host is no longer a viable recovery target for failed members.

To perform this task, you must be the DB2 cluster services administrator.

Procedure

1. As an instance user, perform the following steps:
   a. Perform a member drain operation using the **QUIESCE** parameter of the **db2stop** command, as follows:
      ```
      db2stop member member-id quiesce 30
      ```
      where *member-id* represents the member that you want to put into maintenance mode.
      For more information on quiescing a member, see "Quiescing a member" on page 82.
   b. Before you stop the host, make sure that there are only the members targeted for maintenance on the host. Stop the instance on the member host, as follows:
      ```
      db2stop instance on host-name
      ```
      where *host-name* represents the name of the host of the given member or CF.
2. As the DB2 cluster services administrator, perform the following steps:
   a. Put the cluster manager into maintenance mode on the host, by issuing the following command:
      ```
      DB2DIR/bin/db2cluster -cm -enter -maintenance
      ```
      where *DB2DIR* represents the installation location of your DB2 copy.
   b. Put the cluster file system service on the host into maintenance mode, by issuing the following command:
      ```
      DB2DIR/bin/db2cluster -cfs -enter -maintenance
      ```
      where *DB2DIR* represents the installation location of your DB2 copy.
   c. Reboot the host.

What to do next

1. As the DB2 cluster services administrator, perform the following steps:

a. Perform any maintenance activities planned. For example, make hardware configuration changes. Note that updates to the DB2 pureScale software components require that you put all hosts into maintenance mode.
b. Exit cluster manager maintenance mode on the host by issuing the following command:
```
DB2DIR/bin/db2cluster -cm -exit -maintenance
```
where *DB2DIR* represents the installation location of your DB2 copy.
c. Exit cluster file system maintenance mode on the host by issuing the following command:
```
DB2DIR/bin/db2cluster -cfs -exit -maintenance
```
where *DB2DIR* represents the installation location of your DB2 copy.

2. As an instance user, perform the following steps:
a. Restart the instance on the host by issuing the following command:
```
db2start instance on host-name
```
where *host-name* represents the name of the host of the given member or CF.
b. Restart the member by issuing the following command:
```
db2start member member-id
```
where *member-id* represents the member that you want to put into maintenance mode.

Replacing both cluster caching facilities

You can replace both cluster caching facilities using a rolling upgrade technique which ensures that your DB2 pureScale instance will not experience an outage.

Before you begin

You must have root user authority to perform this task.

Ensure that you know which cluster caching facility is currently the primary, so that you don't inadvertently stop it. As well, the secondary cluster caching facility needs to be in PEER state. You can accomplish both of these items by querying the DB2_CF administrative view, as follows: `SELECT * FROM SYSIBMADM.DB2_CF`

Procedure

Use the following steps to replace both cluster caching facilities (CFs):

1. Stop the secondary cluster caching facility using the command: `db2stop CF CF-identifier` For more information about stopping a cluster caching facility, see "Stopping a cluster caching facility" on page 75.
2. Stop the instance on the secondary cluster caching facility to ensure no other DB2 processes are running, using the command: `db2stop instance on host-name`
3. Drop the cluster caching facility using the command:`db2iupdt -drop -cf host-name instance-name`
4. Add the new host using the command: `db2iupdt -add -cf host-name:net-name instance-name`
5. Find out the ID for the newly added cluster caching facility using the statement: `SELECT * from SYSIBMADM.DB2_CF`

6. Start the newly added cluster caching facility using the command: `db2start CF CF-identifier` For more information about starting a cluster caching facility, see "Starting a cluster caching facility" on page 73.
7. Once the new cluster caching facility (now the secondary cluster caching facility) reaches PEER state, stop the primary cluster caching facility as in steps 1 and 2. DB2 cluster services will fail over the primary role to the secondary cluster caching facility.
8. Repeat steps 3 - 6 to replace the old primary cluster caching facility and restart it as the secondary cluster caching facility.

Adding a disk to the shared file system

After creating your shared file systems, there might be circumstances which require adding additional disks to the file system.

Before you begin

The user ID running this command must own the disks and have read and write access to them. The disks cannot be currently used by any other file system on the local host. To perform this task, you must be either the user ID that created the file system, or the DB2 cluster services administrator.

Procedure

Once the disk is physically available to the hosts, you can add the disk to the file system that requires more space by using the **db2cluster** command. For example:

```
db2cluster -add -filesystem filesystem-name -disk disk-name
```

Results

When this task is completed, the disk is added to the file system, and the stripe set for the file system includes this disk.

What to do next

If you want to ensure that the file system is balanced over all the disks in the file system, perform a rebalance operation.

Removing a disk from the shared file system:

If you decide that your file system does not require all of the disks currently mounted on it, or that a given disk might be better deployed with a file system that requires more storage, you can easily remove a disk by using the **db2cluster** command.

Before you begin

Ensure that there is sufficient space on the remaining disks in the file system to accommodate the files that are on the disk being removed.

About this task

This task removes a disk from the file system and makes it available for use elsewhere.

Restrictions

- The instance owner can perform this task only if the instance owner created the file system; otherwise only the DB2 cluster services can remove a disk from a file system.
- If there is only one disk left in the file system, you cannot remove it using this method. Delete the file system instead, as described in com.ibm.db2.luw.admin.sd.doc/doc/t0056124.dita.
- You cannot remove the disk tiebreaker. To find out if a disk is a tiebreaker, see com.ibm.db2.luw.admin.sd.doc/doc/c0056704.dita.
- You can remove only one disk at a time from a file system.

Procedure

To remove a disk from the shared file system, use the **db2cluster** command:

```
db2cluster -cfs -remove -filesystem filesystem-name -disk disk-name
```

What to do next

Once the disk is removed from your file system, run a rebalance operation to ensure that the file system is balanced over all remaining disks in the file system as described in "Rebalancing a file system."

Rebalancing a file system:

You can rebalance your file system after adding or removing disks. Perform a rebalance operation by using the **db2cluster** command.

Before you begin

Ensure that the file system is mounted before running this command. Because a rebalance operation is file input and output intensive, it is recommended that you run them when there is reduced system activity in the cluster.

About this task

This task rebalances the file system by restriping the data on disk across all disks in the file system. The instance owner can perform this task only if the instance owner created the file system; otherwise only the DB2 cluster services administrator can rebalance a file system.

Procedure

Use the **db2cluster** command to rebalance the file system:

```
db2cluster -rebalance -filesystem filesystem-name
```

Quiescing a member

Certain circumstances might require you to temporarily remove a member from the cluster (for example, maintenance operations).

About this task

In a DB2 pureScale environment, the **db2stop** and **STOP DATABASE MANAGER** commands offer the optional **QUIESCE** parameter. The optional **QUIESCE** parameter allows you to drain all activity against a single member and shut it down. While the member is being drained, automatic client reroute (ACR) and workload balancing will reroute new transactions and new connections to other members.

When a **db2stop QUIESCE** command is issued against a member, applications are subject to these conditions:

- If you specify a timeout value, applications have up to that amount of time to finish their active units of work.
- If no timeout (or a value of -1) is specified, then the server waits indefinitely, until all active transactions and associated connections on the member have ended.
- If you specify a value of 0 (zero) for the timeout value, connections are forced off immediately.

Note: When a stop member quiesce is issued (with no or non-zero timeout), active transactions are marked for deferred connection termination. When a transaction commits or rolls back successfully, the connection will be terminated unless one of these conditions is true:

- The connection uses global variables
- An encrypted password is used
- There is an open WITH HOLD cursor
- Declared temporary tables (DGTT) are used
- A TRANSFORM GROUP is set
- The SESSION AUTHID is changed
- PL/SQL packages or SQL/PL modules are used
- Cursor variables are used
- Sequence values are used
- Created temporary tables (CGTT) with PRESERVE ROWS are used
- Dynamic SQL prepared in a package bound with KEEPDYNAMIC YES. This restriction does not apply when preparing statements in a stored procedure or user-defined function, or when a statement is prepared by an IBM non-embedded API such as CLI/JDBC/ODBC/.NET.

If any of the preceding conditions is true, you must either remove it (such as WITH HOLD cursors), or explicitly stop the connection. In the absence of any of these conditions, the client connection will be terminated at the next transaction end point. If your client (.NET, CLI, JDBC) supports seamless ACR, your connections are automatically rerouted to another member.

Procedure

Run the **db2stop** command or the **STOP DATABASE MANAGER** command and specify the **QUIESCE** parameter.
This example takes member 2 offline and allows 30 minutes for the active workload to finish. After 30 minutes, all applications are forced off the member.

```
db2stop MEMBER 2 QUIESCE 30
```

To bring member 2 back online, issue the **db2start** command against it.

```
db2start MEMBER 2
```

Example

This example uses the db2InstanceStop API to bring member 10 offline.

```
struct sqlca sqlca;     // sqlca to carry the sqlcode
struct db2InstanceStopStruct instanceStopStruct;
struct db2StopOptionsStruct  stopOptions;
```

```
instanceStopStruct.iIsRemote = FALSE; // demo local instance
instanceStopStruct.piRemoteInstName = NULL;
instanceStopStruct.piCommData = NULL; // don't care DAS
instanceStopStruct.piStopOpts = &stopOptions;

stopOptions.iOption = SQLE_QUIESCE; // Member quiesce option
stopOptions.iIsType = TRUE;
stopOptions.iType = DB2_NODE_MEMBER;
stopOptions.iIsNodeNum = TRUE;
stopOptions.iNodeNum = 10;
stopOptions.iQuiesceDeferMinutes = 0; // no explicit timeout

// Finally, invoke the API to shut down the instance
db2InstanceStop(db2Version1010, &instanceStopStruct, &sqlca);
```

This examples uses the db2InstanceStop API to bring member 10 offline and specifies a timeout of 5 minutes.

```
struct sqlca sqlca;     // sqlca to carry the sqlcode
struct db2InstanceStopStruct instanceStopStruct;
struct db2StopOptionsStruct  stopOptions;

instanceStopStruct.iIsRemote = FALSE; // demo local instance
instanceStopStruct.piRemoteInstName = NULL;
instanceStopStruct.piCommData = NULL; // don't care DAS
instanceStopStruct.piStopOpts = &stopOptions;

stopOptions.iOption = SQLE_QUIESCE; // Member quiesce option
stopOptions.iIsType = TRUE;
stopOptions.iType = DB2_NODE_MEMBER;
stopOptions.iIsNodeNum = TRUE;
stopOptions.iNodeNum = 10;
stopOptions.iQuiesceDeferMinutes = 5; // timeout of 5 minutes

// Finally, invoke the API to shut down the instance
db2InstanceStop(db2Version1010, &instanceStopStruct, &sqlca);
```

Putting hosts into maintenance mode

If you are applying software updates to DB2 cluster services, you must put the target host into maintenance mode. You can also put a host into maintenance mode if you want to ensure that members or cluster caching facilities are not restarted on the host when you are making updates to the operating system or hardware on the host.

Before you begin

Any DB2 server processes and any other processes accessing the file systems managed by DB2 cluster services must be shut down before running this command. All instances must be stopped on the host before the host can enter maintenance mode. When the instance is stopped on the host, the host is no longer a viable recovery target for failed members.

To perform this task, you must be the DB2 cluster services administrator.

A host cannot enter maintenance mode if any server processes are still active on the host. Even if you specify the **-force** option with the command, this will only shut down the cluster manager and the shared file system.

In order to keep your DB2 pureScale instance from experiencing an outage, follow these recommendations:

- Do not place the hosts in both of the cluster caching facilities into maintenance mode at the same time.
- Do not place all host with members into maintenance mode at the same time.
- Of the total N hosts in the cluster, (N/2) + 1 hosts must be online and not in maintenance mode.

About this task

The more hosts you have in maintenance mode at the same time, the fewer hosts are available for recovery in the event of an outage or failure. As a result, putting more than one host into maintenance mode at the same time is not recommended.

Procedure

- To place a host in maintenance mode, see "Performing maintenance on a member host" on page 79.
- To enter maintenance mode on all hosts, see "Putting a cluster into maintenance mode."

Putting a cluster into maintenance mode

You can put a cluster into maintenance mode when you are making updates to the operating system or hardware on the hosts in the cluster.

To perform this task, you must be the DB2 cluster services administrator.

Procedure

1. As an instance user, perform the following steps:
   a. Stop the database manager on all hosts by issuing the following command on a single host:

      ```
      su -iname
      db2stop
      exit
      ```

      where *iname* represents the instance owner name.
   b. On each host, stop the DB2 instance by issuing the following command:

      ```
      db2stop instance on hostname
      ```

      where *hostname* represents the host name for a given member or CF, and the `db2stop instance on hostname` command is run for each host in the cluster.
2. As the DB2 cluster services administrator, perform the following steps:
   a. Issue the following command on each resource group to unmount the resource groups:

      ```
      rgreq -o stop resource-group-name
      ```

      where *resource-group-name* represents the name of each resource group returned by running the `lsrg | grep db2mnt` command. You can verify that the resource group is offline by using the `lssam -g resource-group-name` command.
   b. Put the cluster manager into maintenance mode on all hosts by issuing the following command:

      ```
      DB2DIR/bin/db2cluster -cm -enter -maintenance -all
      ```

      where *DB2DIR* represents the installation location of your DB2 copy.

c. Put the cluster file system service on the hosts into maintenance mode by issuing the following command:

```
DB2DIR/bin/db2cluster -cfs -enter -maintenance -all
```

where *DB2DIR* represents the installation location of your DB2 copy.

d. Check the value of the **autoload** configuration parameter by issuing the **mmlsconfig** command:

```
mmlsconfig | grep autoload
```

If the value of the autoload configuration parameter is set to YES, issue the following command to change the value to NO to prevent the nodes from being started automatically:

```
mmchconfig autoload=no -N all
```

e. Stop all hosts.

What to do next

1. As the DB2 cluster services administrator, perform the following steps:
   a. Perform any maintenance activities planned. For example, install a fix pack update, or make network topology changes to your DB2 pureScale environment.
   b. Check the value of the **autoload** configuration parameter by issuing the **mmlsconfig** command:

```
mmlsconfig | grep autoload
```

If the value of the **autoload** configuration parameter is set to NO, issue the following command to change the value to YES:

```
mmchconfig autoload=yes -N all
```

   c. Exit cluster manager maintenance mode on the host by issuing the following command:

```
DB2DIR/bin/db2cluster -cm -exit -maintenance
```

where *DB2DIR* represents the installation location of your DB2 copy.

   d. Ensure that all members and domain are active by issuing the following command:

```
DB2DIR/bin/db2cluster -cm -list -host -state
```

where *DB2DIR* represents the installation location of your DB2 copy.

   e. Reset the resource group state by issuing the following command on each resource group:

```
rgreq -o cancel resource-group-name
```

where *resource-group-name* represents the name of each resource group returned by running the `lsrg | grep db2mnt` command.

   f. Exit cluster file system maintenance mode on the hosts by issuing the following command:

```
DB2DIR/bin/db2cluster -cfs -exit -maintenance -all
```

where *DB2DIR* represents the installation location of your DB2 copy. In case of timeout, check the status of the cluster file system by issuing the `DB2DIR/bin/db2cluster -cfs -list -host -state` command.

2. As an instance user, perform the following steps:
   a. On each host, start the DB2 instance by issuing the following command:

```
db2start instance on hostname
```

where *hostname* represents the host name for a given member or CF, and the `db2start instance on hostname` command is run for each host in the cluster.

b. Start the database manager by issuing the following command:

```
su -iname
db2start
exit
```

where *iname* represents the instance owner name.

Moving a DB2 member or a cluster caching facility

There are a number of reasons to move a DB2 member or a cluster caching facility from one host to another. This task outlines a number of possible scenarios and some additional factors to consider.

About this task

Following a change in a DB2 pureScale environment with a **db2iupdt -add** or **-drop** operation, a backup of the database might be required. This backup must be performed from one of the preexisting instance members.

Restrictions

A maximum of two cluster caching facilities are supported. If you want to move one, you must drop the other one first, then add the second one on a new host. You receive an error if you try and add a third cluster caching facility before first dropping one.

There must always be at least one DB2 member and one cluster caching facility. If you want to move one, you must add the new DB2 member or cluster caching facility on a new host first before dropping the original one. You receive an error if you try to drop the last remaining member or cluster caching facility.

Procedure

- To move the only DB2 member:
  1. Stop the DB2 pureScale instance on all hosts by using the **db2stop** command. This step is necessary because the **db2iupdt -add** and **-drop** commands must run offline.
  2. Add a new member. This addition ensures that there is always at least one DB2 member. For example, add a DB2 member called `Member2` with a netname of `Netname2` to the DB2 pureScale instance `sdinstA`:

     ```
     db2iupdt -add -m Member2:Netname2 sdinstA
     ```

  3. Drop the original DB2 member. For example, drop the member called `Member1` with a netname of `Netname1` from the DB2 pureScale instance `sdinstA`:

     ```
     db2iupdt -drop -m Member1:Netname1 sdinstA
     ```

  4. Back up the database on this DB2 pureScale instance from a member that existed before you started this process if the database is recoverable. Recoverable databases have the **logarchmeth1** or **logarchmeth2** database configuration parameters set to a value other than `OFF`.
- To move one of multiple DB2 members:

1. The **db2iupdt** -add or -drop command must run offline, so stop the instance on all hosts by using the **db2stop** command.
2. Drop the DB2 members you want to move. For example, drop the member Member2 with a netname of Netname2 from the DB2 pureScale instance sdinstA:

   ```
   db2iupdt -drop -m Member2:Netname2 sdinstA
   ```
3. Add a new DB2 member. For example, add Member3 with a netname of Netname3 as a member to the DB2 pureScale instance sdinstA:

   ```
   db2iupdt -add -m Member3:Netname3 sdinstA
   ```
4. Back up the database on this DB2 pureScale instance from a member that existed before you started this process if the database is recoverable. Recoverable databases have the **logarchmeth1** or **logarchmeth2** database configuration parameters set to a value other than OFF.

- To move the only cluster caching facility:
  1. The **db2iupdt** -add or -drop command must run offline, so stop the instance on all hosts by using the **db2stop** command.
  2. Add a new cluster caching facility. This addition ensures that there is always at least one cluster caching facility. For example, add a cluster caching facility called cf2 with a netname of NetCF2 to the DB2 pureScale instance sdinstA:

     ```
     db2iupdt -add -cf cf2:NetCF2 sdinstA
     ```
  3. Drop the initial cluster caching facility. For example, drop the cluster caching facility called cf1 with a netname of NetCF1 from the DB2 pureScale instance sdinstA:

     ```
     db2iupdt -drop -cf cf1:NetCF1 sdinstA
     ```
- To move one of two cluster caching facilities:
  1. The **db2iupdt** -add or -drop command must run offline, so stop the instance on all hosts by using the **db2stop** command.
  2. A maximum of two cluster caching facilities are supported, so you must drop one first. For example, drop a cluster caching facility called cf1 with a netname of NetCF1 from the DB2 pureScale instance sdinstA:

     ```
     db2iupdt -drop -cf cf1:NetCF1 sdinstA
     ```
  3. Add the second cluster caching facility on a different host. For example, add a cluster caching facility called cf2 with a netname of NetCF2 to the DB2 pureScale instance sdinstA:

     ```
     db2iupdt -add -cf cf2:NetCF2 sdinstA
     ```

Recovering from a failed db2iupdt -add or -drop operation in a DB2 pureScale environment

There are a number of reasons why a **db2iupdt** -add or -drop operation might fail, leaving the DB2 pureScale environment topology in an inconsistent state.

If the problem requires a hardware fix (for example, a host failure), finish that task first. Once the hardware is up and running, you can use the **db2iupdt** command with the -fixtopology option to automatically either roll back the previous **db2iupdt** -add operation, or finish the previous **db2iupdt** -drop operation.

If the operation that failed was an attempt to drop a member host with multiple DB2 members, then the **db2iupdt** -fixtopology command will only finish the previous **db2iupdt** -drop of the one specific member that failed. It might be necessary to rerun the original **db2iupdt** -drop command to finish dropping the other DB2 members on the member host.

Clearing resource model alerts

In a DB2 pureScale environment, hardware- or software-related alerts can exist on the status table for members, cluster caching facilities (CFs), and for hosts. In some cases, the alert conditions are transient so the alert field might clear itself; however, in other cases, the alert field remains set until the administrator resolves the problem and manually clears the alert .

About this task

When this command is run, all alerts for the instance on the specified member, cluster caching facility, or host are cleared.

To perform this task, you must be a user in the SYSADM, SYSMAINT, or SYSADM group.

Procedure

To clear alerts for a member, cluster caching facility, or host, issue this command:

```
db2cluster -clear -alert  [-member member-id | -cf cf-id | -host host-name]
```

Results

Once the command completes, the system might or might not take a specific action related to the component that had its alert cleared. In some cases, the member is returned to its home host, but if a restart light failure caused the alert, no action is taken.

Changing the cluster manager quorum type

The DB2 cluster services tiebreaker is set up initially in the installation GUI when the first instance is created. Change the cluster manager quorum type only if a problem occurs with the specified disk or the configuration changes.

Before you begin

- The DB2 pureScale instance must be stopped on all hosts before the quorum type can be changed. The cluster manager must remain online.
- If you are changing to a disk tiebreaker, you can specify the disk in a device path format or as a PVID or WWN number. The disk you specify cannot be used for any other purpose.
- The disk option for the cluster manager can be entered in the form of a device path (for example: /dev/hdisk0), or as a PVID (port virtual local area network identifier) or WWN (world wide name) number.
- To perform this task, you must be the DB2 cluster services administrator.

Procedure

- To change the quorum type to a disk tiebreaker, issue this command:

  ```
  db2cluster -cm -set -tiebreaker -disk diskname
  ```

- To change the quorum type to a majority node set, issue this command:

  ```
  db2cluster -cm -set -tiebreaker -majority
  ```

Changing the shared file system quorum type

The shared file system quorum type is assigned automatically during the setup of the IBM DB2 pureScale Feature. Because the quorum type selected is based upon the number of hosts in the cluster, you must change the quorum type after adding or removing hosts from the DB2 instance.

Before you begin

- The DB2 instance and shared file system cluster must be shut down on all hosts before the quorum type can be changed. If the shared file system cluster is still active, issue a **db2cluster -cfs -stop** command.
- If you are changing to a disk tiebreaker, the disk you specify must be of the device path format (for example: `/dev/hdisk2`) and it must be accessible on all of the nodes. Unlike the disk tiebreaker for the cluster manager, this disk can be reused in a shared file system, or in a disk on an already defined file system.
- To perform this task, you must be the DB2 cluster services administrator.

About this task

There are two quorum types for GPFS™:

- *Disk tiebreaker*: a shared disk that determines which surviving group of hosts in the GPFS cluster has operational quorum
- *Majority node set*: the surviving group of hosts in the GPFS with the majority has operational quorum

The quorum type is automatically chosen by the DB2 installer based upon the rule in Table 7.

Table 7. The relationship between number of hosts in a GPFS cluster and the quorum type

Total amount of hosts in GPFS cluster	GPFS quorum type
Equal or less than 8	Disk tiebreaker
More than 8	Majority Node Set

When the number of hosts in the GPFS increases from eight hosts or decreases from nine hosts, an error message is returned indicating that the quorum type must be changed.

Procedure

- To change the quorum type to a disk tiebreaker, issue this command:
  ```
  db2cluster -cfs -set -tiebreaker -disk diskname
  ```
- To change the quorum type to a majority node set, issue this command:
  ```
  db2cluster -cfs -set -tiebreaker -majority
  ```

Changing the host failure detection time

Fast failure detection is crucial to a DB2 pureScale instance because the faster a host failure is detected, the earlier the recovery for the failure can begin. That said, an aggressive setting for the host failure detection time is not suited for high-latency environments, so users might want to adjust the setting.

Before you begin

- The DB2 instance and the shared file system cluster must be shut down on all hosts before the host failure detection time can be changed. If the shared file system cluster is still active, issue a **db2cluster -cfs -stop -all** command.
- To perform this task you must be the DB2 cluster services administrator.

About this task

You can use the **db2cluster** command to adjust the fast failure detection time for your DB2 pureScale instance. The command specifies how long it takes to detect a host failure or network partition. To determine the current setting use the command:

```
db2cluster -cm -list -HostFailureDetectionTime
```

Procedure

To change the host failure detection time, use the command:

```
db2cluster -cm -set -option HostFailureDetectionTime -value value
```

Results

Once the command completes, the new setting applies to all DB2 pureScale instances in the DB2 cluster services domain.

Part 2. Databases

Chapter 5. 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.

Recommended file systems

DB2 databases run on many of the file systems supported by the platforms the DB2 product runs on.

The file systems you can use depend on whether you want use the IBM DB2 pureScale Feature.

- "DB2 environments without the DB2 pureScale Feature"
- "DB2 pureScale environments"

DB2 environments without the DB2 pureScale Feature

IBM recommends the file systems shown in Table 8 for DB2 for Linux, UNIX, and Windows.

Note: Undocumented limitations might exist for file systems not listed in Table 8. Support for unlisted file systems might be limited.

Table 8. File systems recommended for DB2 for Linux, UNIX, and Windows

Platform	Operating System	File systems recommended
Linux	Red Hat Enterprise Linux (RHEL)	• IBM General Parallel File System[1] (GPFS) • ext3 • VERITAS File System (VxFS) • Network File System (NFS[2]) with IBM N-series • Network File System (NFS[2]) with Network Appliance filers
	SUSE Linux Enterprise Server (SLES)	• GPFS • ext3 • VERITAS File System (VxFS) • NFS[2] with IBM N-series • NFS[2] with Network Appliance filers
UNIX	AIX	• GPFS • Enhanced Journaled File System (JFS2) • NFS[2] with IBM N-series • NFS[2] with Network Appliance filers • VxFS
	HP-UX	• HP JFS[3] (VxFS)
	Solaris	• UNIX File System (UFS) • ZFS • VxFS
Windows	All Windows products	NTFS

Notes:

[1] See http://www-03.ibm.com/systems/clusters/software/gpfs/index.html for additional information about GPFS.

[2] See http://www-01.ibm.com/support/docview.wss?uid=swg21169938 for details on which NFS versions are validated for use on various operating systems.

[3] HP JFS on HP-UX is an OEM version of VxFS.

DB2 pureScale environments

The DB2 pureScale Feature requires the IBM General Parallel File System (GPFS). This file system is installed and configured automatically by the DB2 pureScale Feature installer if it is not already installed. Refer to the installation prerequisites for more information about the specific version of GPFS required. For more information about GPFS, see http://www-03.ibm.com/systems/clusters/software/gpfs/index.html.

Database directories and files

When you create a database, information about the database including default information is stored in a directory hierarchy.

The hierarchical directory structure is created for you. You can specify the location of the structure by specifying a directory path or drive for the **CREATE DATABASE** command; if you do not specify a location, a default location is used.

In the directory that you specify as the database path in the **CREATE DATABASE** command, a subdirectory that uses the name of the instance is created.

Within the instance-name subdirectory, the *partition-global directory* is created. The partition-global directory contains global information associated with your new database. The partition-global directory is named NODE*xxxx*/SQL*yyyyy*, where *xxxx* is the data partition number and *yyyyy* is the database token (numbered >=1).

Under the *partition-global directory*, the *member-specific* directory is created. The member-specific directory contains local database information. The member-specific directory is named MEMBER*xxxx* where *xxxx* is the member number.

- In a DB2 pureScale environment, there is a member-specific directory for each member, called MEMBER0000, MEMBER0001, and so on.
- In a partitioned database environment, member numbers have a one-to-one mapping with their corresponding partition number, therefore there is one NODE*xxxx* directory per member and partition. Member-specific directories are always named MEMBERxxxx and they always reside under the partition-global directory.
- An Enterprise Server Edition environment runs on a single member, and has one member-specific directory, called MEMBER0000.

Partition-global directory

The partition-global directory has the path: *your_instance*/*NODExxxx*/SQLxxxxx.

The partition-global directory contains the following files:

- Global deadlock write-to-file event monitor files that specify either a relative path or no path at all.
- Table space information files.

  The files SQLSPCS.1 and SQLSPCS.2 contain table space information. These files are duplicates of each other for backup purposes.
- Storage group control files.

  The files SQLSGF.1 and SQLSGF.2 contain storage group information associated with the automatic storage feature of a database. These files are duplicates of each other for maintenance and backup purposes. The files are created for a database when you create the database using the **CREATE DATABASE** command or when you upgrade a nonautomatic storage database to DB2 Version 10.1 or later.
- Temporary table space container files.

  The default directory for new containers is instance/NODExxxx/<db-name>. The files are managed locally by each member. The table space file names are made unique for each member by inserting the member number into the file name, for example: /*storage path*/SAMPLEDB/T0000011/C0000000.TMP/SQL00002.MEMBER0001.TDA
- The global configuration file.

The global configuration file, SQLDBCONF, contains database configuration parameters that refer to single, shared resources that must remain consistent across the database. Do not edit this file. To change configuration parameters, use the **UPDATE DATABASE CONFIGURATION** and **RESET DATABASE CONFIGURATION** commands.

- History files.

  The DB2RHIST.ASC history file and its backup DB2RHIST.BAK contain history information about backups, restores, loading of tables, reorganization of tables, altering of a table space, and other changes to a database.

  The DB2TSCHG.HIS file contains a history of table space changes at a log-file level. For each log file, DB2TSCHG.HIS contains information that helps to identify which table spaces are affected by the log file. Table space recovery uses information from this file to determine which log files to process during table space recovery. You can examine the contents of history files in a text editor.

- Logging-related files.

  The global log control files, SQLOGCTL.GLFH.1, SQLOGCTL.GLFH.2, contain recovery information at the database level, for example, information related to the addition of new members while the database is offline and maintaining a common log chain across members. The log files themselves are stored in the LOGSTREAM*xxxx* directories (one for each member) in the partition-global directory.

- Locking files.

  The instance database lock files, SQLINSLK,and SQLTMPLK, help to ensure that a database is used by only one instance of the database manager.

- Automatic storage containers

Member-specific directory

The member-specific directory has the path: /NODE*xxxx*/SQL*xxxx*/MEMBER*xxxx*

This directory contains objects associated with the first database created, and subsequent databases are given higher numbers: SQL00002, and so on. These subdirectories differentiate databases created in this instance on the directory that you specified in the **CREATE DATABASE** command.

The database directory contains the following files:

- Buffer pool information files.

  The files SQLBP.1 and SQLBP.2 contain buffer pool information. These files are duplicates of each other for backup purposes.

- Local event monitor files.

- Logging-related files.

  The log control files, SQLOGCTL.LFH.1, its mirror copy SQLOGCTL.LFH.2, and SQLOGMIR.LFH, contain information about the active logs. In a DB2 pureScale environment, each member has its own log stream and set of local LFH files, which are stored in each member-specific directory.

  Tip: Map the log subdirectory to disks that you are not using for your data. By doing so, you might restrict disk problems to your data or the logs, instead of having disk problems for both your data and the logs. Mapping the log subdirectory to disks that you are not using for your data can provide a substantial performance benefit because the log files and database containers do not compete for movement of the same disk heads. To change the location of the log subdirectory, use the **newlogpath** database configuration parameter.

- The local configuration file.

  The local SQLDBCONF file contains database configuration information. Do not edit this file. To change configuration parameters, use the **UPDATE DATABASE CONFIGURATION** and **RESET DATABASE CONFIGURATION** commands.

At the same time a database is created, a detailed deadlocks event monitor is also created. In an Enterprise Server Edition environment and in partitioned database environments, the detailed deadlocks event monitor files are stored in the database directory of the catalog node. In a DB2 pureScale environment, the detailed deadlocks event monitor files are stored in the partition-global directory. When the event monitor reaches its maximum number of files to output, it will deactivate and a message is written to the notification log. This prevents the event monitor from using too much disk space. Removing output files that are no longer needed allows the event monitor to activate again on the next database activation.

Additional information for SMS database directories in non-automatic storage databases

In non-automatic storage databases, the SQLT* subdirectories contain the default System Managed Space (SMS) table spaces:

- SQLT0000.0 subdirectory contains the catalog table space with the system catalog tables.
- SQLT0001.0 subdirectory contains the default temporary table space.
- SQLT0002.0 subdirectory contains the default user data table space.

Each subdirectory or container has a file created in it called SQLTAG.NAM. This file marks the subdirectory as being in use so that subsequent table space creation does not attempt to use these subdirectories.

In addition, a file called SQL*.DAT stores information about each table that the subdirectory or container contains. The asterisk (*) is replaced by a unique set of digits that identifies each table. For each SQL*.DAT file there might be one or more of the following files, depending on the table type, the reorganization status of the table, or whether indexes, LOB, or LONG fields exist for the table:

- SQL*.BKM (contains block allocation information if it is an MDC or ITC table)
- SQL*.LF (contains LONG VARCHAR or LONG VARGRAPHIC data)
- SQL*.LB (contains BLOB, CLOB, or DBCLOB data)
- SQL*.XDA (contains XML data)
- SQL*.LBA (contains allocation and free space information about SQL*.LB files)
- SQL*.INX (contains index table data)
- SQL*.IN1 (contains index table data)
- SQL*.DTR (contains temporary data for a reorganization of an SQL*.DAT file)
- SQL*.LFR (contains temporary data for a reorganization of an SQL*.LF file)
- SQL*.RLB (contains temporary data for a reorganization of an SQL*.LB file)
- SQL*.RBA (contains temporary data for a reorganization of an SQL*.LBA file)

Database configuration file

A *database configuration file* is created for each database. This file is called SQLDBCON prior to Version 8.2, and SQLDBCONF in Version 8.2 and later. The creation of this file is done for you.

This file contains values for various *configuration parameters* that affect the use of the database, such as:

- Parameters specified or used when creating the database (for example, database code page, collating sequence, DB2 database release level)
- Parameters indicating the current state of the database (for example, backup pending flag, database consistency flag, rollforward pending flag)
- Parameters defining the amount of system resources that the operation of the database might use (for example, buffer pool size, database logging, sort memory size).

Note: If you edit the db2system, SQLDBCON (prior to Version 8.2), or SQLDBCONF (Version 8.2 and later) file using a method other than those provided by the DB2 database manager, you might make the database unusable. Therefore, do not change these files using methods other than those documented and supported by the database manager.

Performance Tip: Many of the configuration parameters come with default values, but might need to be updated to achieve optimal performance for your database. By default, the Configuration Advisor is invoked as part of the **CREATE DATABASE** command so that the initial values for some parameters are already configured for your environment.

For multi-partition databases: When you have a database that is distributed across more than one database partition, the configuration file should be the same on all database partitions. Consistency is required since the query compiler compiles distributed SQL statements based on information in the local node configuration file and creates an access plan to satisfy the needs of the SQL statement. Maintaining different configuration files on database partitions could lead to different access plans, depending on which database partition the statement is prepared.

Node directory

The database manager creates the *node directory* when the first database partition is cataloged.

To catalog a database partition, use the **CATALOG NODE** command.

Note: In a DB2 pureScale environment, it is not necessary to run the CATALOG NODE command, because the DB2 pureScale Feature acts as a single partition.

To list the contents of the local node directory, use the **LIST NODE DIRECTORY** command.

The node directory is created and maintained on each database client. The directory contains an entry for each remote database partition server having one or more databases that the client can access. The DB2 client uses the communication end point information in the node directory whenever a database connection or instance attachment is requested.

The entries in the directory also contain information about the type of communication protocol to be used to communicate from the client to the remote database partition. Cataloging a local database partition creates an alias for an instance that resides on the same computer.

Local database directory

A *local database directory* file exists in each path (or "drive" for Windows operating systems) in which a database has been defined. This directory contains one entry for each database accessible from that location.

Each entry contains:

- The database name provided with the **CREATE DATABASE** command
- The database alias name (which is the same as the database name, if an alias name is not specified)
- A comment describing the database, as provided with the **CREATE DATABASE** command
- The name of the root directory for the database
- Other system information.

System database directory

A *system database directory* file exists for each instance of the database manager, and contains one entry for each database that has been cataloged for this instance.

Databases are implicitly cataloged when the **CREATE DATABASE** command is issued and can also be explicitly cataloged with the **CATALOG DATABASE** command.

For each database created, an entry is added to the directory containing the following information:

- The database name provided with the **CREATE DATABASE** command
- The database alias name (which is the same as the database name, if an alias name is not specified)
- The database comment provided with the **CREATE DATABASE** command
- The location of the `local database directory`
- An indicator that the database is *indirect*, which means that it resides on the current database manager instance
- Other system information.

On Linux and UNIX platforms and in a partitioned database environment, you must ensure that all database partitions always access the same system database directory file, `sqldbdir`, in the `sqldbdir` subdirectory of the home directory for the instance. Unpredictable errors can occur if either the system database directory or the system intention file `sqldbins` in the same `sqldbdir` subdirectory are symbolic links to another file that is on a shared file system.

Creating node configuration files

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

About this task

To enable database partitioning, the `db2nodes.cfg` file must be located in the `sqllib` subdirectory of the home directory for the instance before you start the database manager. This file contains configuration information for all database partitions in an instance, and is shared by all database partitions for that instance.

Windows considerations

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

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

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

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

Tokens are delimited by blanks. The variables are:

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

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

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

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

  `NODE` *nnnn*

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

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

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

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

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

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

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

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

Table 9. Database partition number example table.

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

You can update the `db2nodes.cfg` file using an editor of your choice. (The exception is: an editor should not be used on Windows.) You must be careful, however, to protect the integrity of the information in the file, as database partitioning requires that the node configuration file is locked when you issue **START DBM** and unlocked after **STOP DBM** ends the database manager. The **START DBM** command can update the file, if necessary, when the file is locked. For example, you can issue **START DBM** with the **RESTART** option or the **ADD DBPARTITIONNUM** option.

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

Changing node and database configuration files

To update the database configuration file, run the **AUTOCONFIGURE** command with the appropriate options.

About this task

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.

If you plan to change any database partition groups (adding or deleting database partitions, or moving existing database partitions), the node configuration file must be updated. If you plan to change the database, review the values for the configuration parameters. You can adjust some values periodically as part of the ongoing changes made to the database that are based on how it is used.

Note: If you modify any parameters, the values are not updated until:

- For database parameters, the first new connection to the database after all applications are disconnected
- For database manager parameters, the next time that you stop and start the instance

In most cases, the values recommended by the Configuration Advisor provide better performance than the default values because they are based on information about your workload and your own particular server. However, the values are designed to improve the performance of, though not necessarily optimize, your database system. Think of the values as a starting point on which you can make further adjustments to obtain optimized performance.

In Version 9.1, the Configuration Advisor is automatically invoked when you create a database. To disable this feature, or to explicitly enable it, use the **db2set** command before creating the database. Examples:

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

See "Automatic features" on page 21 for other features that are enabled by default.

Procedure

To change database or database manager configuration parameters:

- Use the Configuration Advisor.
  - From the command line, use the **AUTOCONFIGURE** command.
  - From a client application, call the db2AutoConfig API.
- From IBM Data Studio, right-click the instance or database to open the task assistant to change the database manager or database configuration parameters.
- From the command line, use the **UPDATE DATABASE MANAGER CONFIGURATION** and **UPDATE DATABASE CONFIGURATION** commands.

  For example, to update individual parameters in the database manager configuration, enter:

  ```
  UPDATE DBM CFG USING config_keyword value
  ```

  To update individual parameters in the database configuration, enter:

  ```
  UPDATE DB CFG FOR database_alias
  USING config_keyword value
  ```

  You can update one or more *config_keyword value* combinations in a single command.

  Most changes to the database manager configuration file become effective only after they are loaded into memory. For a server configuration parameter, this occurs during the running of the **START DATABASE MANAGER** command. For a client configuration parameter, this occurs when the application is restarted.
- From a client application, call the db2CfgSet API.

What to do next

To view or print the current database manager configuration parameters, use the **GET DATABASE MANAGER CONFIGURATION** command.

Database recovery log

A *database recovery log* keeps a record of all changes made to a database, including the addition of new tables or updates to existing ones.

This log is made up of a number of *log extents*, each contained in a separate file called a *log file*.

The database recovery log can be used to ensure that a failure (for example, a system power outage or application error) does not leave the database in an inconsistent state. In case of a failure, the changes already made but not committed are rolled back, and all committed transactions, which might not have been physically written to disk, are redone. These actions ensure the integrity of the database.

Space requirements for database objects

Estimating the size of database objects is an imprecise undertaking. Overhead caused by disk fragmentation, free space, and the use of variable length columns makes size estimation difficult, because there is such a wide range of possibilities for column types and row lengths.

After initially estimating your database size, create a test database and populate it with representative data. Then use the db2look utility to generate data definition statements for the database.

When estimating the size of a database, the contribution of the following must be considered:

- System catalog tables
- User table data
- Long field (LF) data
- Large object (LOB) data
- XML data
- Index space
- Log file space
- Temporary work space

Also consider the overhead and space requirements for the following:

- The local database directory file
- The system database directory file
- The file management overhead required by the operating system, including:
  - File block size
  - Directory control space

Space requirements for log files

Space requirements for log files vary depending on your needs and on configuration parameter settings.

You require 56 KB of space for log control files. You also require at least enough space for your active log configuration, which you can calculate as

$$(logprimary + logsecond) \times (logfilsiz + 2) \times 4096$$

where:

- `logprimary` is the number of primary log files, defined in the database configuration file

- logsecond is the number of secondary log files, defined in the database configuration file; in this calculation, logsecond cannot be set to -1. (When logsecond is set to -1, you are requesting an infinite active log space.)
- logfilsiz is the number of pages in each log file, defined in the database configuration file
- 2 is the number of header pages required for each log file
- 4096 is the number of bytes in one page.

Rollforward recovery

If the database is enabled for rollforward recovery, special log space requirements might be considered:

- With the **logarchmeth1** configuration parameter set to LOGRETAIN, the log files are archived in the log path directory. The online disk space eventually fills up, unless you move the log files to a different location.
- With the **logarchmeth1** configuration parameter set to USEREXIT, DISK, or VENDOR, a user exit program moves the archived log files to a different location. Extra log space is still required to allow for:
  - Online archived log files that are waiting to be moved by the user exit program
  - New log files being formatted for future use
- You can reduce the cost of storing your archived log files if you enable compression on these files.
  - For example, if the **logarchmeth1** configuration parameter is set to DISK, TSM, or VENDOR, and you set the **logarchcompr1** configuration parameter to ON, archived log files are compressed and you reduce the cost of storing these files. If you enable compression dynamically, existing archived log files already stored are not compressed. Compression starts with the current active log file when compression is enabled.

Circular logging

If the database is enabled for circular logging, the result of this formula is all the space is allocated for logging; that is, more space is not allocated, and you do not receive insufficient disk space errors for any of your log files.

Infinite logging

If the database is enabled for infinite logging (that is, you set the **logsecond** configuration parameter to -1), the **logarchmeth1** configuration parameter must be set to a value other than OFF or logretain to enable archive logging. The database manager keeps at least the number of active log files specified by the **logprimary** configuration parameter in the log path, therefore, you must not use the value of -1 for the **logsecond** configuration parameter in the formula shown previously. Ensure that you provide extra disk space to allow for the delay caused by archiving log files.

Mirroring log paths

If you are mirroring the log path, you must double the estimated log file space requirements.

Currently committed

If queries return the currently committed value of the data, more log space is required for logging the first update of a data row during a transaction when the **cur_commit** configuration parameter is not set to DISABLED. Depending on the size of the workload, the total log space used can vary

significantly. This scenario affects the log I/O required for any workload, the amount of active log space required, and the amount of log archive space required.

Note: Setting the **cur_commit** configuration parameter to DISABLED, maintains the same behavior as in previous releases, and results in no changes to the log space required.

Lightweight Directory Access Protocol (LDAP) directory service

A directory service is a repository of resource information about multiple systems and services within a distributed environment; and it provides client and server access to these resources.

Clients and servers would use the directory service to find out how to access other resources. Information about these other resources in the distributed environment must be entered into the directory service repository.

Lightweight Directory Access Protocol (LDAP) is an industry standard access method to directory services. Each database server instance will publish its existence to an LDAP server and provide database information to the LDAP directory when the databases are created. When a client connects to a database, the catalog information for the server can be retrieved from the LDAP directory. Each client is no longer required to store catalog information locally on each computer. Client applications search the LDAP directory for information required to connect to the database.

Note: Publishing of the database server instance to the LDAP server is not an automatic process, but must be done manually by the administrator.

As an administrator of a DB2 system, you can establish and maintain a directory service. The directory service is made available to the DB2 database manager through Lightweight Directory Access Protocol (LDAP) directory services. To use LDAP directory services, there must first exist an LDAP server that is supported by the DB2 database manager so that directory information can be stored there.

Note: When running in a Windows domain environment, an LDAP server is already available because it is integrated with the Windows Active Directory. As a result, every computer running Windows can use LDAP.

An LDAP directory is helpful in an enterprise environment where it is difficult to update local directory catalogs on each client computer because of the large number of clients. In this situation, you should consider storing your directory entries in an LDAP server so that maintaining catalog entries is done in one place: on the LDAP server.

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.

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.

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.

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 10. 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 defined in the backup image	Uses database path of existing database	Uses storage paths defined in the backup image
`RESTORE DATABASE TEST2 TO X:`	`X:`	Uses storage paths defined in the backup image	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 defined in the backup image	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 11. 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

Cataloging databases

When you create a database, it is automatically cataloged in the system database directory file. You might also use the **CATALOG DATABASE** command to explicitly catalog a database in the system database directory file.

About this task

You can use the **CATALOG DATABASE** command to catalog a database with a different alias name, or to catalog a database entry that was previously deleted using the **UNCATALOG DATABASE** command.

Although databases are cataloged automatically when a database is created, you still might need to catalog the database. When you do so, the database must exist.

By default, directory files, including the database directory, are cached in memory using the Directory Cache Support (**dir_cache**) configuration parameter. When directory caching is enabled, a change made to a directory (for example, using a

CATALOG DATABASE or **UNCATALOG DATABASE** command) by another application might not become effective until your application is restarted. To refresh the directory cache used by a command line processor session, issue the **TERMINATE** command.

In a partitioned database, a cache for directory files is created on each database partition.

In addition to the application level cache, a database manager level cache is also used for internal, database manager look-up. To refresh this "shared" cache, issue the **db2stop** and **db2start** commands.

Procedure

- To catalog a database with a different alias name using the command line processor, use the **CATALOG DATABASE** command.

  For example, the following command line processor command catalogs the PERSON1 database as HUMANRES:

  ```
  CATALOG DATABASE personl AS humanres
    WITH "Human Resources Database"
  ```

  Here, the system database directory entry has HUMANRES as the database alias, which is different from the database name (PERSON1).
- To catalog a database in the system database directory from a client application, call the sqlecadb API.
- To catalog a database on an instance other than the default using the command line processor, use the **CATALOG DATABASE** command.

  In the following example, connections to database B are to INSTNC_C. The instance instnc_c must already be cataloged as a local node before attempting this command.

  ```
  CATALOG DATABASE b as b_on_ic AT NODE instnc_c
  ```

  Note: The **CATALOG DATABASE** command is also used on client nodes to catalog databases that reside on database server computers.

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.

Connecting to distributed relational databases

Distributed relational databases are built on formal requester-server protocols and functions.

An *application requester* supports the application end of a connection. It transforms a database request from the application into communication protocols suitable for use in the distributed database network. These requests are received and processed by a *database server* at the other end of the connection. Working together, the application requester and the database server handle communication and location considerations, so that the application can operate as if it were accessing a local database.

An application process must connect to a database manager's application server before SQL statements that reference tables or views can be executed. The CONNECT statement establishes a connection between an application process and its server.

There are two types of CONNECT statements:

- CONNECT (Type 1) supports the single database per unit of work (Remote Unit of Work) semantics.
- CONNECT (Type 2) supports the multiple databases per unit of work (Application-Directed Distributed Unit of Work) semantics.

The DB2 call level interface (CLI) and embedded SQL support a connection mode called *concurrent transactions*, which allows multiple connections, each of which is an independent transaction. An application can have multiple concurrent connections to the same database.

The application server can be local to or remote from the environment in which the process is initiated. An application server is present, even if the environment is not using distributed relational databases. This environment includes a local directory that describes the application servers that can be identified in a CONNECT statement.

The application server runs the bound form of a static SQL statement that references tables or views. The bound statement is taken from a package that the database manager has previously created through a bind operation.

For the most part, an application connected to an application server can use statements and clauses that are supported by the application server's database manager. This is true even if an application is running through the application requester of a database manager that does *not* support some of those statements and clauses.

Remote unit of work for distributed relational databases

The *remote unit of work facility* provides for the remote preparation and execution of SQL statements.

An application process at computer system "A" can connect to an application server at computer system "B" and, within one or more units of work, execute any number of static or dynamic SQL statements that reference objects at "B". After ending a unit of work at B, the application process can connect to an application server at computer system C, and so on.

Most SQL statements can be remotely prepared and executed, with the following restrictions:

- All objects referenced in a single SQL statement must be managed by the same application server.
- All of the SQL statements in a unit of work must be executed by the same application server.

At any given time, an application process is in one of four possible *connection states*:

- Connectable and connected

  An application process is connected to an application server, and CONNECT statements can be executed.

  If implicit connect is available:

  – The application process enters this state when a CONNECT TO statement or a CONNECT without operands statement is successfully executed from the connectable and unconnected state.

  – The application process might enter this state from the implicitly connectable state if any SQL statement other than CONNECT RESET, DISCONNECT, SET CONNECTION, or RELEASE is issued.

  Whether or not implicit connect is available, this state is entered when:

  – A CONNECT TO statement is successfully executed from the connectable and unconnected state.

  – A COMMIT or ROLLBACK statement is successfully issued, or a forced rollback occurs from the unconnectable and connected state.

- Unconnectable and connected

  An application process is connected to an application server, but a CONNECT TO statement cannot be successfully executed to change application servers. The application process enters this state from the connectable and connected state when it executes any SQL statement other than the following: CONNECT TO, CONNECT with no operand, CONNECT RESET, DISCONNECT, SET CONNECTION, RELEASE, COMMIT, or ROLLBACK.

- Connectable and unconnected

  An application process is not connected to an application server. CONNECT TO is the only SQL statement that can be executed; otherwise, an error (SQLSTATE 08003) is raised.

  Whether or not implicit connect is available, the application process enters this state if an error occurs when a CONNECT TO statement is issued, or an error occurs within a unit of work, causing the loss of a connection and a rollback. An error that occurs because the application process is not in the connectable state, or because the server name is not listed in the local directory, does not cause a transition to this state.

If implicit connect is not available:

- The application process is initially in this state
- The CONNECT RESET and DISCONNECT statements cause a transition to this state.

- Implicitly connectable (if implicit connect is available).

  If implicit connect is available, this is the initial state of an application process. The CONNECT RESET statement causes a transition to this state. Issuing a COMMIT or ROLLBACK statement in the unconnectable and connected state, followed by a DISCONNECT statement in the connectable and connected state, also results in this state.

Availability of implicit connect is determined by installation options, environment variables, and authentication settings.

It is not an error to execute consecutive CONNECT statements, because CONNECT itself does not remove the application process from the connectable state. It is, however, an error to execute consecutive CONNECT RESET statements. It is also an error to execute any SQL statement other than CONNECT TO, CONNECT RESET, CONNECT with no operand, SET CONNECTION, RELEASE, COMMIT, or ROLLBACK, and then to execute a CONNECT TO statement. To avoid this error, a CONNECT RESET, DISCONNECT (preceded by a COMMIT or ROLLBACK statement), COMMIT, or ROLLBACK statement should be executed before the CONNECT TO statement.

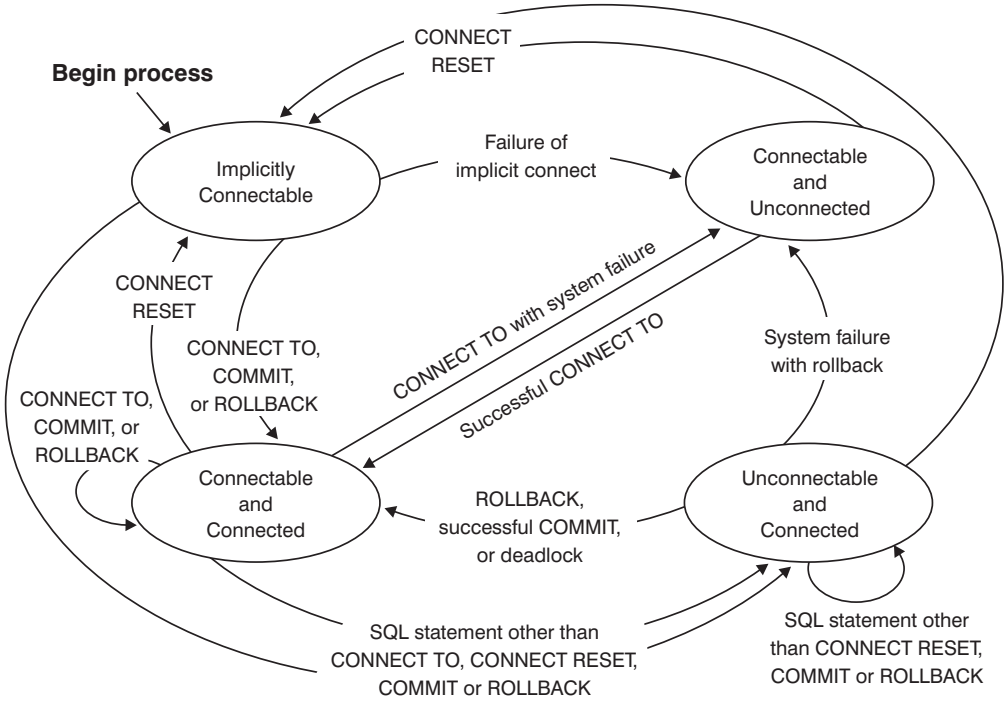


Figure 4. Connection State Transitions If Implicit Connect Is Available

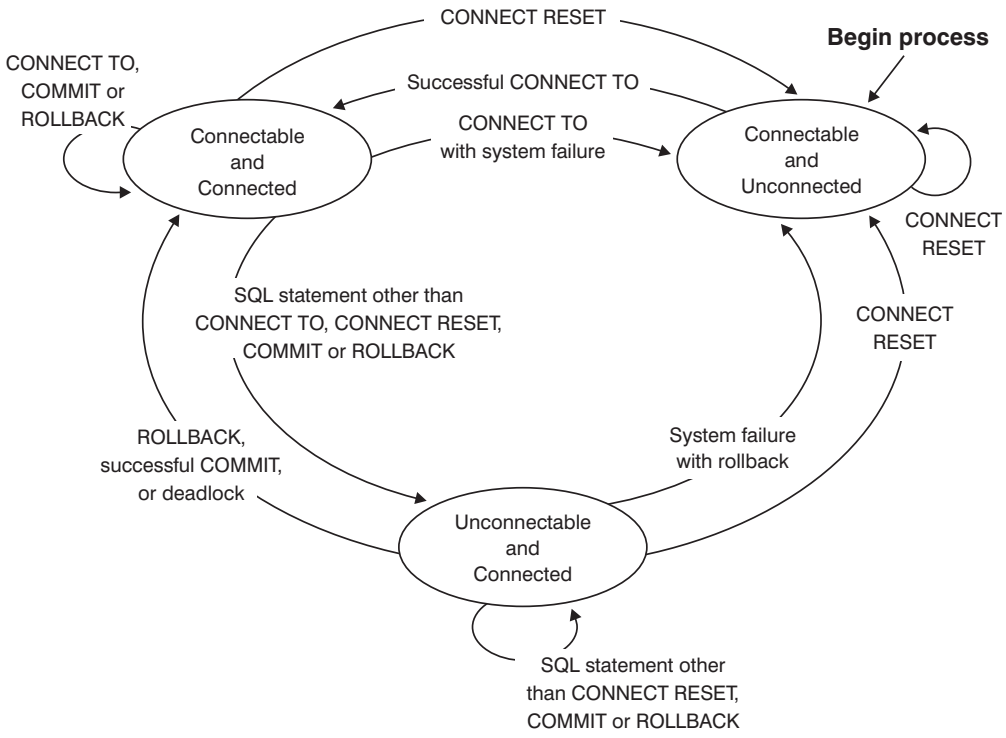


Figure 5. Connection State Transitions If Implicit Connect Is Not Available

Application-directed distributed unit of work

The *application-directed distributed unit of work facility* provides for the remote preparation and execution of SQL statements.

An application process at computer system "A" can connect to an application server at computer system "B" by issuing a CONNECT or a SET CONNECTION statement. The application process can then execute any number of static and dynamic SQL statements that reference objects at "B" before ending the unit of work. All objects referenced in a single SQL statement must be managed by the same application server. However, unlike the remote unit of work facility, any number of application servers can participate in the same unit of work. A commit or a rollback operation ends the unit of work.

An application-directed distributed unit of work uses a type 2 connection. A *type 2* connection connects an application process to the identified application server, and establishes the rules for application-directed distributed units of work.

A type 2 application process:

- Is always connectable
- Is either in the connected state or in the unconnected state
- Has zero or more connections.

Each connection of an application process is uniquely identified by the database alias of the application server for the connection.

An individual connection always has one of the following connection states:

- current and held
- current and release-pending
- dormant and held
- dormant and release-pending

A type 2 application process is initially in the unconnected state, and does not have any connections. A connection is initially in the current and held state.

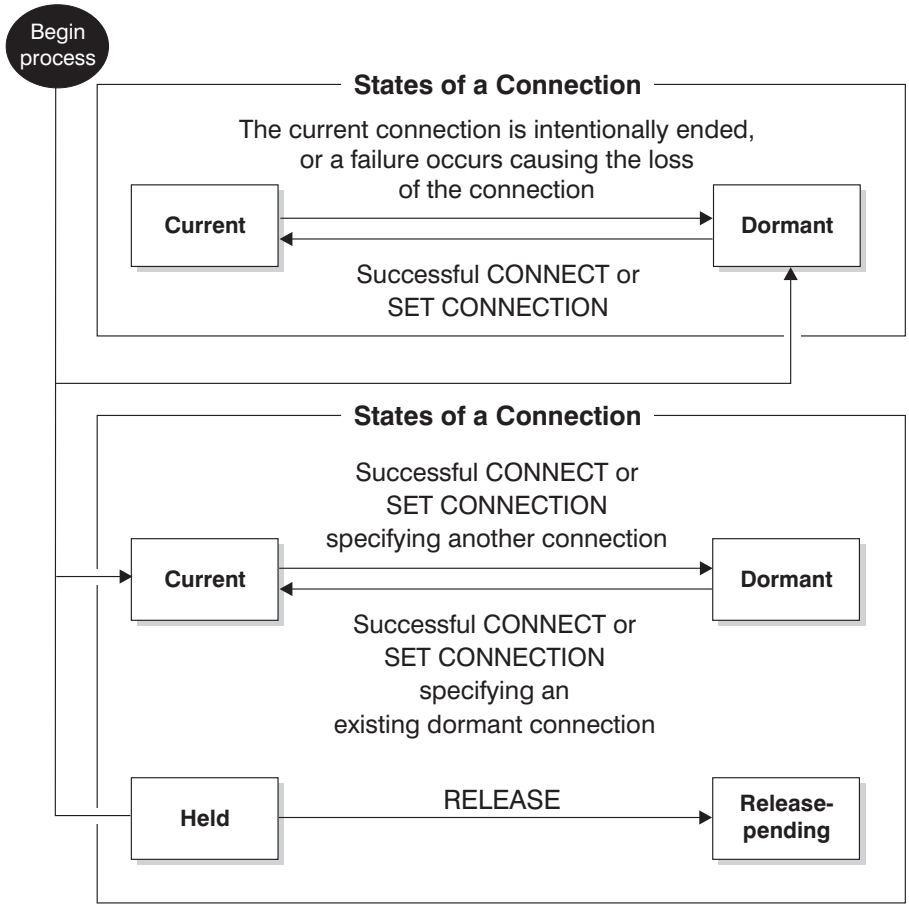


Figure 6. Application-Directed Distributed Unit of Work Connection State Transitions

Application process connection states

There are certain rules that apply to the execution of a CONNECT statement.

The following rules apply to the execution of a CONNECT statement:

- A context cannot have more than one connection to the same application server at the same time.
- When an application process executes a SET CONNECTION statement, the specified location name must be an existing connection in the set of connections for the application process.
- When an application process executes a CONNECT statement, and the SQLRULES(STD) option is in effect, the specified server name must *not* be an existing connection in the set of connections for the application process. For a description of the SQLRULES option, see "Options that govern unit of work semantics" on page 127.

If an application process has a current connection, the application process is in the *connected* state. The CURRENT SERVER special register contains the name of the application server for the current connection. The application process can execute SQL statements that refer to objects managed by that application server.

An application process that is in the unconnected state enters the connected state when it successfully executes a CONNECT or a SET CONNECTION statement. If there is no connection, but SQL statements are issued, an implicit connect is made, provided the **DB2DBDFT** environment variable is set with the name of a default database.

If an application process does not have a current connection, the application process is in the *unconnected* state. The only SQL statements that can be executed are CONNECT, DISCONNECT ALL, DISCONNECT (specifying a database), SET CONNECTION, RELEASE, COMMIT, ROLLBACK, and local SET statements.

An application process in the *connected state* enters the *unconnected state* when its current connection intentionally ends, or when an SQL statement fails, causing a rollback operation at the application server and loss of the connection. Connections end intentionally following the successful execution of a DISCONNECT statement, or a COMMIT statement when the connection is in release-pending state. (If the DISCONNECT precompiler option is set to AUTOMATIC, all connections end. If it is set to CONDITIONAL, all connections that do not have open WITH HOLD cursors end.)

Connection states

There are two types of connection states: “held and release-pending states” and “current and dormant states”.

If an application process executes a CONNECT statement, and the server name is known to the application requester but is not in the set of existing connections for the application process: (i) the current connection is placed into the *dormant connection state*, the server name is added to the set of connections, and the new connection is placed into both the *current connection state* and the *held connection state*.

If the server name is already in the set of existing connections for the application process, and the application is precompiled with the SQLRULES(STD) option, an error (SQLSTATE 08002) is raised.

Held and release-pending states. The RELEASE statement controls whether a connection is in the held or the release-pending state. The *release-pending* state means that a disconnect is to occur at the next successful commit operation. (A rollback has no effect on connections.) The *held* state means that a disconnect is *not* to occur at the next commit operation.

All connections are initially in the held state and can be moved to the release-pending state using the RELEASE statement. Once in the release-pending state, a connection cannot be moved back to the held state. A connection remains in release-pending state across unit of work boundaries if a ROLLBACK statement is issued, or if an unsuccessful commit operation results in a rollback operation.

Even if a connection is not explicitly marked for release, it might still be disconnected by a commit operation if the commit operation satisfies the conditions of the DISCONNECT precompiler option.

Current and dormant states. Regardless of whether a connection is in the held state or the release-pending state, it can also be in the current state or the dormant state. A connection in the *current* state is the connection being used to execute SQL statements while in this state. A connection in the *dormant* state is a connection that is not current.

The only SQL statements that can flow on a dormant connection are COMMIT, ROLLBACK, DISCONNECT, or RELEASE. The SET CONNECTION and CONNECT statements change the connection state of the specified server to current, and any existing connections are placed or remain in dormant state. At any point in time, only one connection can be in current state. If a dormant connection becomes current in the same unit of work, the state of all locks, cursors, and prepared statements is the same as the state they were in the last time that the connection was current.

When a connection ends

When a connection ends, all resources that were acquired by the application process through the connection, and all resources that were used to create and maintain the connection are de-allocated. For example, if the application process executes a RELEASE statement, any open cursors are closed when the connection ends during the next commit operation.

A connection can also end because of a communications failure. If this connection is in current state, the application process is placed in unconnected state.

All connections for an application process end when the process ends.

Customizing an application environment using the connect procedure

The connect procedure provides you a way to allow applications in your environment to implicitly execute a specific procedure upon connection. This procedure can allow you to customize an application environment to a database from a central point of control.

For example, in the connect procedure you can set special registers such as CURRENT_PATH to non-default values by invoking the SET CURRENT PATH statement. This new CURRENT_PATH value will now be the effective default CURRENT_PATH for all applications.

Any procedure created in the database that conforms to the naming and parameter restrictions can be used as the connect procedure for that database. The customization logic is provided by you in the form of a procedure created in the same database and is allowed to do any of the usual actions of a procedure such as issue SQL statements.

The **connect_proc** database configuration parameter specifies the connect procedure to be used for all connections to the database. Update the **connect_proc** parameter to set the name of the connect procedure and enable it. A database connection is required to update a non-zero length value of the **connect_proc** parameter. After the **connect_proc** parameter is set, the session authorization ID of any new connection must have EXECUTE privilege on the specified connect procedure either directly or indirectly through one of its associated groups, roles, or PUBLIC.

The connect procedure is implicitly executed on the server at the end of successful connection processing and before processing any subsequent requests on the same connection. After the connect procedure runs successfully, the database manager commits any changes made by the connect procedure. If the connect procedure fails, any changes made by the connect procedure are rolled back and the connection attempt fails with an error.

Note: Any changes made to a special register in the connect procedure are reflected in the resulting session even after the procedure finishes.

Important: Any error returned by the connection procedure will fail an attempted connection. The error returned by execution of the connect procedure is returned to the application. If you want to modify the connect procedure and fix the error, you must unset the **connect_proc** parameter to allow connections to succeed until the problem is fixed.

Recommendations for connect procedure

To avoid problems with your connect procedure, ensure that your connect procedure complies with the following recommendations:

- Keep the connect procedure logic simple.

  Using a connect procedure affects the performance of CONNECT commands for every connection by introducing additional processing. The performance impact can be significant if the procedure is inefficient or experiences delays such as lock contention.

  Ensure that the procedure is well tested before establishing it as a connect procedure.
- Avoid accessing objects in the connect procedure that will be dropped or altered.

  If a dependent object in the connect procedure is dropped or privileges to access dependent objects are revoked, the connect procedure might fail. An error returned from the procedure can block new connections to the database from being established based on the logic of your procedure.
- Avoid calling another procedure from the connect procedure.

  Procedures called by the connect procedure can be dropped or altered, unlike the connect procedure itself. If procedures called by the connect procedure are invalidated or dropped, the connect procedure might fail. An error returned from the connect procedure can block new connections to the database from being established based on the logic of your procedure. Also, note that special registers changed in procedures that are called from the connect procedure do not change the special registers of the calling environment (as opposed to special registers changed in the connect procedure itself which do take effect in the application).
- Avoid specifying the COMMIT ON RETURN clause in the connect procedure.

  An internal commit is processed after the implicit call of connect procedure. If the clause COMMIT ON RETURN YES is specified, the database manager processes multiple commit calls that can affect performance. Specifying COMMIT ON RETURN NO has no effect on connect procedure processing.
- Free all resources and close all cursors before exiting the connect procedure.

  Applications cannot access any resources left open (such as WITH HOLD cursors) by the connect procedure. The resources held by the connect procedure after the commit is processed can be freed only when the application finishes.
- Grant EXECUTE privilege to PUBLIC for the connect procedure.

A connect procedure is not intended to control database access. Access control is done by granting database authorities to users.

- Avoid using different values for the **connect_proc** parameter for different database partitions.

  Using different connect procedures for various database partitions can produce inconsistent application behavior depending on the database partition to which users are connected to. It also makes the database environment more complex and difficult to manage.

Usage notes for connect procedure

Connect procedure has the following restrictions and limitations:

- You cannot create a procedure with the same name as a connect procedure while the **connect_proc** parameter is set.
- Only a procedure with exactly zero parameters can be used as a connect procedure. No other procedure sharing the same two-part name can exist in the database as long as the **connect_proc** parameter is set.
- The connect procedure name (both schema and procedure name) can only contain the following characters:
  - A-Z
  - a-z
  - _ (underscore)
  - 0-9

  In addition, the schema and procedure name need to follow the rules of an ordinary identifier.
- You cannot drop or alter the connect procedure while the **connect_proc** parameter is set.

  To alter or drop the connect procedure, change the **connect_proc** parameter to null or the name of a different procedure.
- A connect procedure cannot use client information fields set by the sqleseti API or the SQLSetConnectAttr CLI functions.

  The special register for these fields contains their default server value before the connect procedure runs. The client information fields or SQL special register set by calling the sqleseti API, SQLSetConnectAttr CLI function, or SQLSetEnvAttr CLI function (for example, CLIENT USERID, CLIENT ACCTNG, CLIENT APPLNAME, and CLIENT WRKSTNNAME) are not yet updated when the connect procedure runs.
- The client information fields or SQL special register set by calling the sqleseti API, the SQLSetConnectAttr CLI function, or the SQLSetEnvAttr CLI function, IBM Data Server Driver for JDBC and SQLJ method set ClientAccountingInformation take precedence and override the special register values set in the connect procedure.
- Only special registers that are set directly by the connect procedure will remain set after returning from the connect procedure. The nested routine calls within the connect procedure do not change the settings of the special registers in the calling environment.

Examples of implementing connect procedure

The following examples show you some samples of connect procedure and how the connect procedure is enabled in the database:

Example 1

1. Define an SQL procedure NEWTON.CONNECTPROC to set a special register based on SESSION_USER.

```
CREATE PROCEDURE NEWTON.CONNECTPROC ( )
READS SQL DATA
LANGUAGE SQL
   BEGIN

   --set the special register based on session user id
      CASE SESSION_USER
        WHEN 'USERA' THEN
         SET CURRENT LOCALE LC_TIME 'fr_FR';

        WHEN 'USERB' THEN
         SET CURRENT LOCALE LC_TIME 'de_DE';
   ELSE
         SET CURRENT LOCALE LC_TIME 'au_AU';

   END CASE;

   END %
```

   This procedure establishes a setting for the CURRENT LOCALE LC_TIME special register with special case values for users USERA and USERB.

2. Grant EXECUTE privilege on the connect procedure to the group PUBLIC:

```
GRANT EXECUTE ON PROCEDURE NEWTON.CONNECTPROC TO PUBLIC
```

   .

3. Update the **connect_proc** parameter to indicate that this new procedure is to be invoked for any new connection:

```
db2 update db cfg using connect_proc "NEWTON.CONNECTPROC"
```

   The NEWTON.CONNECTPROC connect procedure is now automatically invoked for any subsequent CONNECT request for a new connection. The special register CURRENT LOCALE LC_TIME is set based on SESSION USER.

Example 2

1. Set up and invoke a procedure for new connections in order to customize their initial special register values.

```
CREATE PROCEDURE MYSCHEMA.CONNECTPROC
  (   )
  EXTERNAL NAME 'parts!connectproc'
  DBINFO
  READS SQL DATA
  LANGUAGE C
  PARAMETER STYLE SQL
```

   This procedure reads from a database table, MYSCHEMA.CONNECTDEFAULTS, to determine what values to set in the CURRENT SCHEMA, CURRENT PATH, and CURRENT QUERY OPTIMIZATION special registers based on the groups associated with the authorization ID of the new connection. It also sets the value of the global variable, MYSCHEMA.SECURITY_SETTING, based on information in the same table.

2. Grant EXECUTE privilege on the connect procedure to the group PUBLIC:

```
GRANT EXECUTE ON PROCEDURE MYSCHEMA.CONNECTPROC TO PUBLIC
```

.

3. Update the **connect_proc** parameter to indicate that this new procedure is to be invoked for any new connection:

   ```
   db2 update db cfg using connect_proc "MYSCHEMA.CONNECTPROC"
   ```

   The MYSCHEMA.CONNECTPROC connect procedure is now automatically invoked for any subsequent CONNECT request for a new connection.

Options that govern unit of work semantics

The semantics of type 2 connection management are determined by a set of precompiler options. These options are summarized in the following list, with default values indicated by bold and underlined text.

- CONNECT (**1** | 2). Specifies whether CONNECT statements are to be processed as type 1 or type 2.
- SQLRULES (**DB2** | STD). Specifies whether type 2 CONNECTs are to be processed according to the DB2 rules, which allow CONNECT to switch to a dormant connection, or the SQL92 Standard rules, which do not allow this.
- DISCONNECT (**EXPLICIT** | CONDITIONAL | AUTOMATIC). Specifies what database connections are to be disconnected when a commit operation occurs:
  - Those that have been explicitly marked for release by the SQL RELEASE statement (EXPLICIT)
  - Those that have no open WITH HOLD cursors, and those that are marked for release (CONDITIONAL)
  - All connections (AUTOMATIC).
- SYNCPOINT (**ONEPHASE** | TWOPHASE | NONE). Specifies how COMMITs or ROLLBACKs are to be coordinated among multiple database connections. This option is ignored, and is included for backwards compatibility only.
  - Updates can only occur against one database in the unit of work, and all other databases are read-only (ONEPHASE). Any update attempts to other databases raise an error (SQLSTATE 25000).
  - A transaction manager (TM) is used at run time to coordinate two-phase COMMITs among those databases that support this protocol (TWOPHASE).
  - Does not use a TM to perform two-phase COMMITs, and does not enforce single updater, multiple reader (NONE). When a COMMIT or a ROLLBACK statement is executed, individual COMMITs or ROLLBACKs are posted to all databases. If one or more ROLLBACKs fail, an error (SQLSTATE 58005) is raised. If one or more COMMITs fail, another error (SQLSTATE 40003) is raised.

To override any of the previously listed options at run time, use the **SET CLIENT** command or the sqlesetc application programming interface (API). Their current settings can be obtained using the **QUERY CLIENT** command or the sqleqryc API. Note that these are not SQL statements; they are APIs defined in the various host languages and in the command line processor (CLP).

Data representation considerations

Different systems represent data in different ways. When data is moved from one system to another, data conversion must sometimes be performed.

Products supporting DRDA® automatically perform any necessary conversions at the receiving system.

To perform conversions of numeric data, the system needs to know the data type and how it is represented by the sending system. Additional information is needed to convert character strings. String conversion depends on both the code page of the data and the operation that is to be performed on that data. Character conversions are performed in accordance with the IBM Character Data Representation Architecture (CDRA). For more information about character conversion, see the *Character Data Representation Architecture: Reference & Registry* (SC09-2190-00) manual.

Viewing the local or system database directory files

Use the **LIST DATABASE DIRECTORY** command to view the information associated with the databases that you have on your system.

Before you begin

Before viewing either the local or system database directory files, you must have previously created an instance and a database.

Procedure

- To see the contents of the local database directory file, issue the following command:

  ```
  LIST DATABASE DIRECTORY ON location
  ```

  where *location* specifies the location of the database.
- To see the contents of the system database directory file, issue the **LIST DATABASE DIRECTORY** command *without* specifying the location of the database directory file.

Dropping databases

Dropping a database can have far-reaching effects, because this action deletes all its objects, containers, and associated files. The dropped database is removed (uncataloged) from the database directories.

Procedure

- To drop a database by using the command line, enter: `DROP DATABASE name`
- To drop a database from a client application, call the sqledrpd API.
- To drop a database at a specified database partition server, call the sqledpan API.
- To drop a database using IBM Data Studio, right-click the database and select the task assistant to drop the database.

Example

The following command deletes the database SAMPLE:

```
DROP DATABASE SAMPLE
```

Note: If you drop the SAMPLE database and find that you need it again, you can re-create it.

Dropping aliases

When you drop an alias, its description is deleted from the catalog, any packages, and cached dynamic queries that reference the alias are invalidated. All views and triggers dependent on the alias are marked inoperative.

Procedure

To drop aliases, from the command line, issue the DROP statement:

```
DROP ALIAS employee-alias
```

Chapter 6. Database partitions

A *database partition* is a part of a database that consists of its own data, indexes, configuration files, and transaction logs. A database partition is sometimes called a node or a database node. A partitioned database environment is a database installation that supports the distribution of data across database partitions.

Chapter 7. 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 135.
- 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)" on page 138.

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 pools in a DB2 pureScale environment

In a DB2 pureScale environment, the cluster caching facility provides a common group buffer pool (GBP) that is shared by all members. Each member also manages its own set of local buffer pools (LBPs).

The GBP is a single buffer pool that supports all DB2 page sizes and that all members can use. Members cache pages in their own LBPs and use the GBP to maintain page consistency between members. LBPs of different page sizes can exist on each member (such as 4K, 8K, 16K, or 32K).

The GBP stores two types of information, directory entries and data elements. Directory entries store metadata information about buffer pool pages, and data elements store page data. The ratio between directory entries and data elements is automatically adjusted by DB2 for Linux, UNIX, and Windows. The GBP memory size is defined by the **cf_gbp_sz** configuration parameter.

DB2 buffer pool service

Because there are LBPs on each member and a GBP that is shared by all members, multiple copies of the same page can exist in more than one buffer pool. Global concurrency and coherency control for accessing the pages, for making changes to a page, and for propagating changes to other members are handled by the DB2 buffer pool service. The service also handles the I/O of data in buffer pools, including the writing of pages in the GBP to disk.

GBP-dependency

A buffer pool page that needs to be accessed by different members in a DB2 pureScale environment is GBP-dependent. For these dependent pages, the GBP coordinates copies of the page in different buffer pools. A page that only one member has access to is not GBP-dependent and exists only in a member LBP. In a DB2 pureScale environment, the temporary table space is not shared between different members, so that any buffer pool pages for the temporary table space are not GBP-dependent.

P-lock control the access of buffer pool pages in a DB2 pureScale environment for updating and reading a version of a page. Unlike the logical lock (such as a row lock or a table lock) that is owned by a particular transaction, the P-locks that control access to buffer pool pages are owned by members of the cluster. The following P-locks are used:

- To update a page, a member must hold the P-lock in X mode.
- To read the latest version of a page, the member must hold the P-lock in S mode.

To read a consistent version, but not necessary the latest version of the page, no P-lock is required. DB2 for Linux, UNIX, and Windows decides internally which type of read is used when accessing a page.

GBP-coherency

When a buffer pool page is dependent on the GBP, it might exist on disk, in the GBP, in a LBP on multiple members, or in a combination of these. The following protocol rules coordinate the coherency of multiple page copies:

- When a member fetches a page into its LBP, it registers it with the GBP.
- When trying to fetch a page into a LBP, a member checks the GBP first and reads the content of the page from disk only if the page does not exist in the GBP (a version of the page that exists in the GBP is never older than the version of the page on disk).
- When a member has an update lock on a page (P-lock in X mode), the page in the LBP of the member can be a newer version than the page in the GBP.
- Before a member releases the update lock (or downgrades the P-lock level), the GBP is updated with the newer version of the page.
- A modified page is written to the GBP when the transaction that modifies it ends, either through a commit or a rollback.
- A page can also be written to the GBP through page negotiation before a transaction ends when another member requests a P-lock to read the latest version or to update the page. When another member requests a P-lock, the lock conflict causes the lock owning member to write a modified page to GBP so that it can release or downgrade the P-lock, after which the lock can be claimed by the requesting member.

GBP control

The total amount of memory to be used by the GBP is controlled by the **cf_gbp_sz** database configuration parameter. The GBP is allocated when the database is first activated on a member, if the GBP does not already exist on the CF. The GBP is deallocated when the CF is stopped, when the database is dropped or consistently shutting down in the entire cluster, or during a database restore operation.

Castout writes pages from the GBP to disk and is coordinated between members. Castout is similar to page cleaning in LBPs and fulfills two functions:

- To write out dirty pages to disk and to ensure there are enough clean directory entries and data elements to use for new page registration and write.
- To maintain a specific recovery window by ensuring that no pages in the GBP are older than a certain age. This reduces the number of log records that must be replayed in the case of a recovery.

If necessary, you can control castout behavior with the **softmax** database configuration parameter. In a DB2 pureScale environment, this parameter determines how many pages must be cast out from the GBP to disk during each work phase.

LBP control

Local buffer pool configuration is controlled through DDL statements. The member through which a DDL statement is issued acts as the coordinator responsible for distributing the statement to other members in the cluster for local execution, and coordinates the overall execution. The behavior of a LBP DDL statement execution in a DB2 pureScale instance has notable differences in comparison to an instance that are not in a DB2 pureScale environment. In a DB2 pureScale environment, not all members (other than the coordinator) with which the LBP is defined must be available or have the database activated for a DDL statement to be successful. For members who are currently unavailable (for example, due to scheduled maintenance) or who do not have the database activated, the DDL statement is not processed by them. The DDL execution continues normally. When the transaction is committed, the buffer pool changes will be updated in the on-disk buffer pool files by the coordinator for all applicable members, including the ones who did not process the statement. These members apply the committed buffer pool changes the next time they activate the database. However, if any active member fails to run the DDL statement, due to an out of memory condition or other errors, the statement is rolled back and an error is returned to the user or client application.

Buffer pool monitor elements

You can review a number of monitoring elements specific to the GBP and LBP to monitor the overall performance of the DB2 pureScale Feature. There are monitor elements specific to the database configuration parameter **cf_gbp_sz**. For more information about viewing memory usage levels for the cluster caching facility see the topic "MON_GET_CF table function - Get cluster caching facility metrics".

There are also a number of monitoring elements for tracking the number of physical page reads, logical page reads, and invalid pages for the GBP and LBP. For a list of monitoring elements for the DB2 pureScale Feature, see "New and changed monitor elements".

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 8. 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 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 7 on page 144 illustrates the relationship between tables and table spaces within a database, and the containers associated with that database.

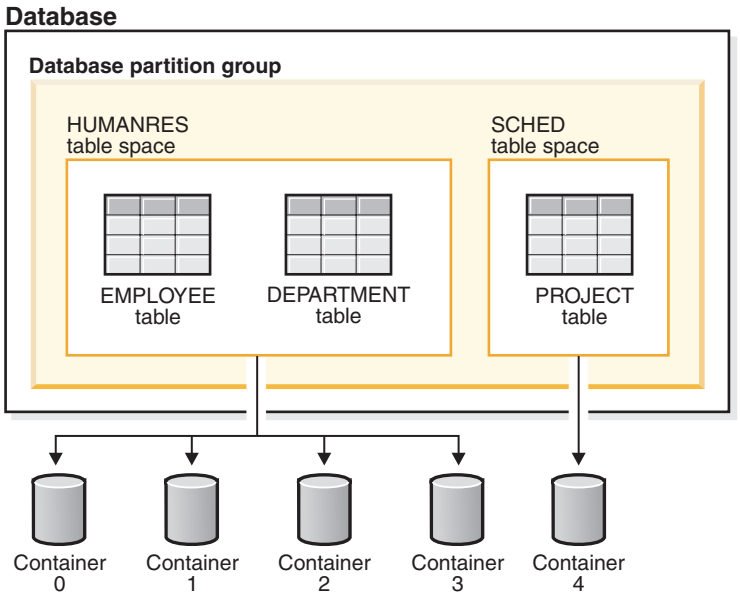


Figure 7. Table spaces and tables in a database

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 8 on page 145 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.

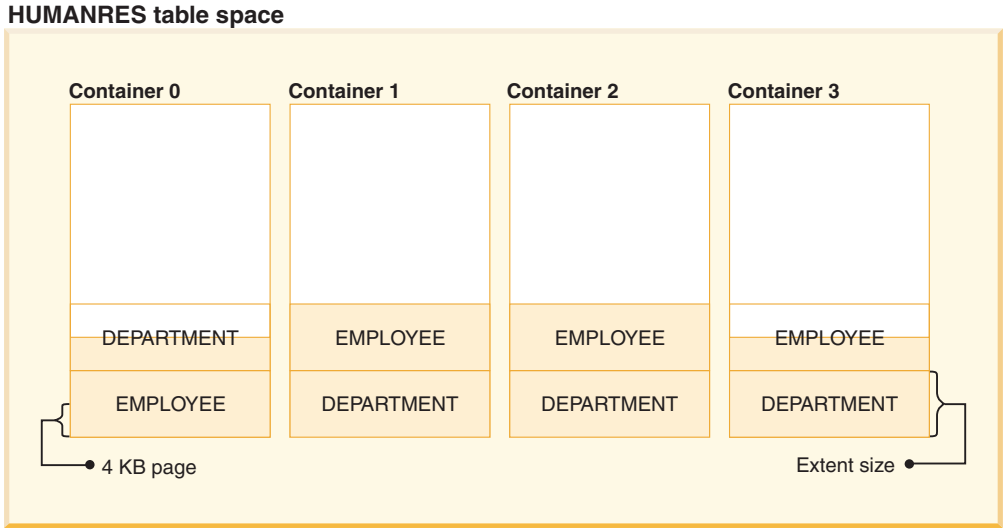


Figure 8. Containers and extents in a table space

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.

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.

Table space considerations for the DB2 pureScale Feature

A DB2 pureScale environment requires specific types of table spaces.

Table space type support

A DB2 pureScale environment supports only the use of automatic storage type table spaces. Prior to migration from an environment other than a DB2 pureScale environment to a DB2 pureScale environment, there must be no system managed space (SMS) tables spaces, regular database managed space (DMS) table spaces, or automatic storage hybrid table spaces. If CREATE TABLESPACE command is issued on a non-automatic storage type in a DB2 pureScale environment, an error is returned (SQL1419N).

Temporary table space support

In a DB2 pureScale configuration, the storage paths for temporary table spaces are defined within the clustered file system. These temporary table spaces are managed locally by each member. The container path is shared by all members, and table space files in that directory are made unique for each member. A member identification number is associated with each file path name.

Transient table space states

In a DB2 pureScale environment, crash recovery after a member failure can take place in parallel during other members' normal runtime activity. These transient states must remain set to block other incompatible operations from starting until member crash recovery (MCR) of that failed member has occurred. There are two types of transient table space states. The first type can be cleared anytime after the member's failure but before recovery starts. As a result of this, if SQLB_BACKUP_IN_PROGRESS, SQLB_DISABLE_PENDING or SQLB_MOVE_IN_PROGRESS are set by the failed member they will remain set until, either:

1. MCR for this database is initiated, or
2. Another member needs to load that table space into its in-memory cache.

The other type of transient table space state needs to remain set until the end of MCR to protect the operations that were running at the time of the failure (for example, a table reorg and certain load commands).

As a result of this, if SQLB_REORG_IN_PROGRESS, or SQLB_LOAD_IN_PROGRESS are set by the failed member they remain set until MCR is completed successfully.

Issuing table space DDL requests

In a partitioned database environment, if the host or DB2 instance of a database partition is down, a table space DDL SQL transaction will fail and the statement will be rolled back. In the case where the host or DB2 instance is up but the database infrastructure is not activated there, the DDL request will implicitly activate the database to process the request. These general behaviors differ in a DB2 pureScale environment as follows:

1. **CREATE TABLESPACE**, **ALTER TABLESPACE**, and **DROP TABLESPACE** operations do not require all members to be up to be completed successfully.
2. If the database infrastructure is not activated on a member, a table space DDL request will not perform implicit activation.

Table spaces and storage management

Table spaces can be set up in different ways, depending on how you want them to use available storage. You can have the operating system manage allocations of space, or you can have the database manager allocate space for your data, based on parameters you specify. Or you can create table spaces that allocate storage automatically.

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

System managed space

In an SMS (System Managed Space) table space, the operating system's file system manager allocates and manages the space where the table is stored. Storage space is allocated on demand.

The SMS storage model consists of files representing database objects; for example, each table has at least one physical file associated with it. When you set up the table space, you decide the location of the files by creating containers. Each container in an SMS table space is associated with an absolute or relative directory name. Each of these directories can be located on a different physical storage device or file system. The database manager controls the names of files created for objects in each container, and the file system is responsible for managing them. By controlling the amount of data written to each file, the database manager distributes the data evenly across the table space containers.

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*

How space is allocated

In an SMS table space, space for tables is allocated on demand. The amount of space that is allocated is dependent on the setting of the **multipage_alloc** database configuration parameter. If this configuration parameter is set to YES (the default), then a full extent (typically made up of two or more pages) is allocated when space is required. Otherwise, space is allocated one page at a time.

Multi-page file allocation affects only the data and index portions of a table. This means that the files used for long data (LONG VARCHAR, LONG VAR GRAPHIC), large objects (LOBs) are not extended one extent at a time.

Note: Multipage file allocation is not applicable to temporary table spaces that use system managed space.

When all space in a single container in an SMS table space is consumed, the table space is considered full, even if space remains in other containers. Unlike DMS table spaces, containers cannot be added to an SMS table space after it is created. Add more space to the underlying file system to provide more space to the SMS container.

Planning SMS table spaces

When considering the use of SMS table spaces, you must consider two factors:

- **The number of containers the table space will need.** When you create an SMS table space, you must specify the number of containers that you want your table space to use. It is important to identify all the containers you want to use, because you cannot add or delete containers after an SMS table space is created. The one exception to this is in a partitioned database environment; when a new database partition is added to the database partition group for an SMS table space, the ALTER TABLESPACE statement can be used to add containers to the new database partition.

  The maximum size of the table space can be estimated by the formula:

  $n \times maxFileSystemSize$

  where n is the number of containers and *maxFileSystemSize* represents the maximum file system size supported by the operating system.

  This formula assumes that each container is mapped to a distinct file system, and that each file system has the maximum amount of space available, and that each file system is of the same size. In practice, this might not be the case, and the maximum table space size might be much smaller. There are also SQL limits on the size of database objects, which might affect the maximum size of a table space.

  Attention: The path you specify for the SMS table space must not contain any other files or directories.

- **The extent size for the table space.** The *extent size* is the number of pages that the database manager writes to a container before using a different container. The extent size can only be specified when the table space is created. Because it cannot be changed later, it is important to select an appropriate value for the extent size.

  If you do not specify the extent size when creating a table space, the database manager creates the table space using the default extent size, defined by the **dft_extent_sz** database configuration parameter. This configuration parameter is

initially set based on information provided when the database is created. If the value for **dft_extent_sz** is not specified for the **CREATE DATABASE** command, the default extent size is set to 32.

Containers and extent size

To choose appropriate number of containers and the extent size for the table space, you must understand:

- **The limitation that your operating system imposes on the size of a logical file system**. For example, some operating systems have a 2 GB limit. Therefore, if you want a 64 GB table object, you will need at least 32 containers on this type of system. When you create the table space, you can specify containers that reside on different file systems and, as a result, increase the amount of data that can be stored in the database.
- **How the database manager manages the data files and containers associated with a table space.** The first table data file (by convention, `SQL00002.DAT`) is created in one of the table space containers. The database manager determines which one, based on an algorithm that takes into account the total number of containers together with the table identifier. This file is allowed to grow to the extent size. After it reaches this size, the database manager writes data to `SQL00002.DAT` in the next container. This process continues until all of the containers contain `SQL00002.DAT` files, at which time the database manager returns to the starting container. This process, known as *striping*, continues through the container directories until a container becomes full (SQL0289N), or no more space can be allocated from the operating system (disk full error). Striping applies to the block map files (`SQL`*`nnnnn`*`.BKM`), to index objects, as well as other objects used to store table data. If you choose to implement disk striping along with the striping provided by the database manager, the extent size of the table space and the strip size of the disk should be identical.

  Note: The SMS table space is deemed to be full as soon as any one of its containers is full. Thus, it is important to have the same amount of space available to each container.

  SMS table spaces are defined using the **MANAGED BY SYSTEM** option on the **CREATE DATABASE** command, or on the CREATE TABLESPACE statement.

Database managed space

In a DMS (database managed space) table space, the database manager controls the storage space. Unlike SMS table spaces, storage space is pre-allocated on the file system based on container definitions that you specify when you create the DMS table space.

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*.

The DMS storage model consists of a limited number of files or devices where space is managed by the database manager. You decide which files and devices to use when creating containers, and you manage the space for those files and devices.

A DMS table space containing user defined tables and data can be defined as a *large* (the default) or *regular* table space that stores any table data or index data.

The maximum size of a regular table space is 512 GB for 32 KB pages. The maximum size of a large table space is 64 TB. See "SQL and XML limits" in the *SQL Reference* for the maximum size of regular table spaces for other page sizes.

There are two options for containers when working with DMS table spaces: files and raw devices. When working with file containers, the database manager allocates the entire container at table space creation time. A result of this initial allocation of the entire table space is that the physical allocation is typically, but not guaranteed to be, contiguous even though the file system is doing the allocation. When working with raw device containers, the database manager takes control of the entire device and always ensures the pages in an *extent* are contiguous. (An *extent* is defined as the number of pages that the database manager writes to a container before using a different container.)

Planning DMS table spaces

When designing your DMS table spaces and containers, you should consider the following:

- The database manager uses striping to ensure an even distribution of data across all containers. This writes the data evenly across all containers in the table space, placing the extents for tables in round-robin fashion across all containers. DB2 striping is recommended when writing data into multiple containers. If you choose to implement disk striping along with DB2 striping, the extent size of the table space and the strip size of the disk should be identical.
- Unlike SMS table spaces, the containers that make up a DMS table space are not required to be the same size; however, this is not normally recommended, because it results in uneven striping across the containers, and sub-optimal performance. If any container is full, DMS table spaces use available free space from other containers.
- Because space is pre-allocated, it must be available before the table space can be created. When using device containers, the device must also exist with enough space for the definition of the container. Each device can have only one container defined on it. To avoid wasted space, the size of the device and the size of the container should be equivalent. For example, if the device is has a storage capacity equivalent to 5000 pages, and the device container is defined to be 3000 pages, 2000 pages on the device will not be usable.
- By default, one extent in every container is reserved for additional required space. Only full extents are used, so for optimal space management, you can use the following formula to determine an appropriate size to use when allocating a container:

  $extent\_size * (n + 1)$

  where *extent_size* is the size of each extent in the table space, and *n* is the number of extents that you want to store in the container.
- The minimum size of a DMS table space is five extents.
  - Three extents in the table space are reserved for overhead:
  - At least two extents are required to store any user table data. (These extents are required for the regular data for one table, and not for any index, long field or large object data, which require their own extents.)

  Attempting to create a table space smaller than five extents will result in an error (SQL1422N).
- Device containers must use logical volumes with a "character special interface," not physical volumes.

- You can use files instead of devices with DMS table spaces. The default table space attribute - NO FILE SYSTEM CACHING in Version 9.5 allows files to perform close to devices with the advantage of not requiring to set up devices. For more information, see "Table spaces without file system caching" on page 180.
- If your workload involves LOBs or LONG VARCHAR data, you might derive performance benefits from file system caching.

  Note: LOBs and LONG VARCHARs are not buffered by the database manager's buffer pool.
- Some operating systems allow you to have physical devices greater than 2 GB in size. You should consider dividing the physical device into multiple logical devices, so that no container is larger than the size allowed by the operating system.

When working with DMS table spaces, you should consider associating each container with a different disk. This allows for a larger table space capacity and the ability to take advantage of parallel I/O operations.

The CREATE TABLESPACE statement creates a new table space within a database, assigns containers to the table space, and records the table space definition and attributes in the catalog. When you create a table space, the extent size is defined as a number of contiguous pages. Only one table or object, such as an index, can use the pages in any single extent. All objects created in the table space are allocated extents in a logical table space address map. Extent allocation is managed through space map pages.

The first extent in the logical table space address map is a header for the table space containing internal control information. The second extent is the first extent of *space map pages* (SMP) for the table space. SMP extents are spread at regular intervals throughout the table space. Each SMP extent is a bit map of the extents from the current SMP extent to the next SMP extent. The bit map is used to track which of the intermediate extents are in use.

The next extent following the SMP is the object table for the table space. The object table is an internal table that tracks which user objects exist in the table space and where their first extent map page (EMP) extent is located. Each object has its own EMPs which provide a map to each page of the object that is stored in the logical table space address map. Figure 9 on page 153 shows how extents are allocated in a logical table space address map.

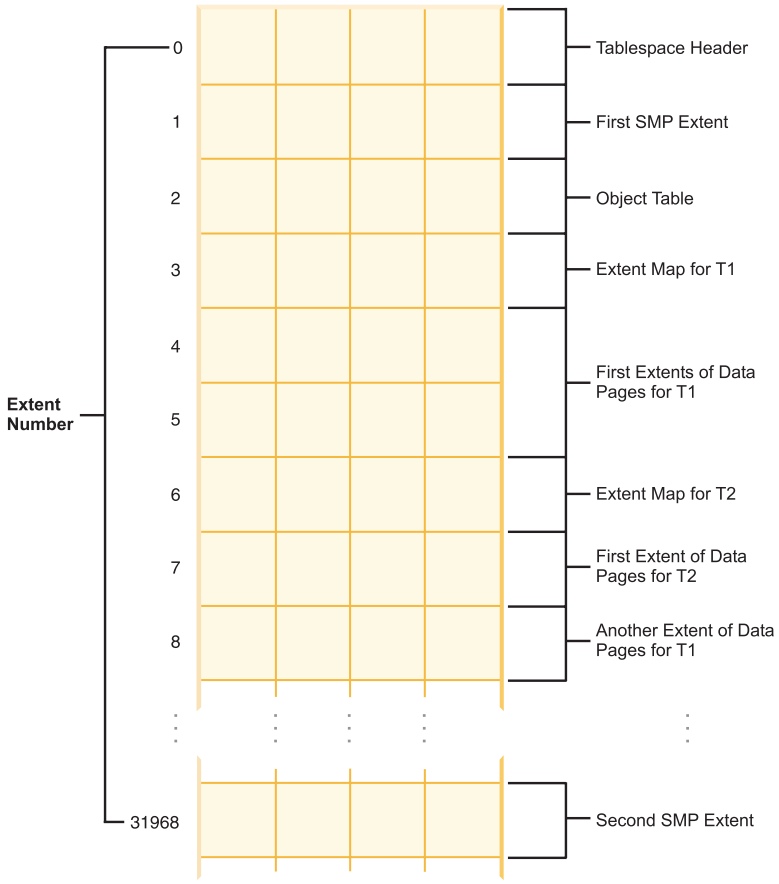


Figure 9. Logical table space address map

Table space maps for database-managed table spaces:

A table space map is the database manager's internal representation of a DMS table space that describes the logical to physical conversion of page locations in a table space. This topic describes why a table space map is useful, and where the information in a table space map comes from.

In a partitioned database, pages in a DMS table space are logically numbered from 0 to (*N*-1), where *N* is the number of usable pages in the table space.

The pages in a DMS table space are grouped into extents, based on the extent size, and from a table space management perspective, all object allocation is done on an extent basis. That is, a table might use only half of the pages in an extent but the whole extent is considered to be in use and owned by that object. By default, one extent is used to hold the container tag, and the pages in this extent cannot be used to hold data. However, if the **DB2_USE_PAGE_CONTAINER_TAG** registry variable is turned on, only one page is used for the container tag.

Figure 10 on page 154 shows the logical address map for a DMS table space.

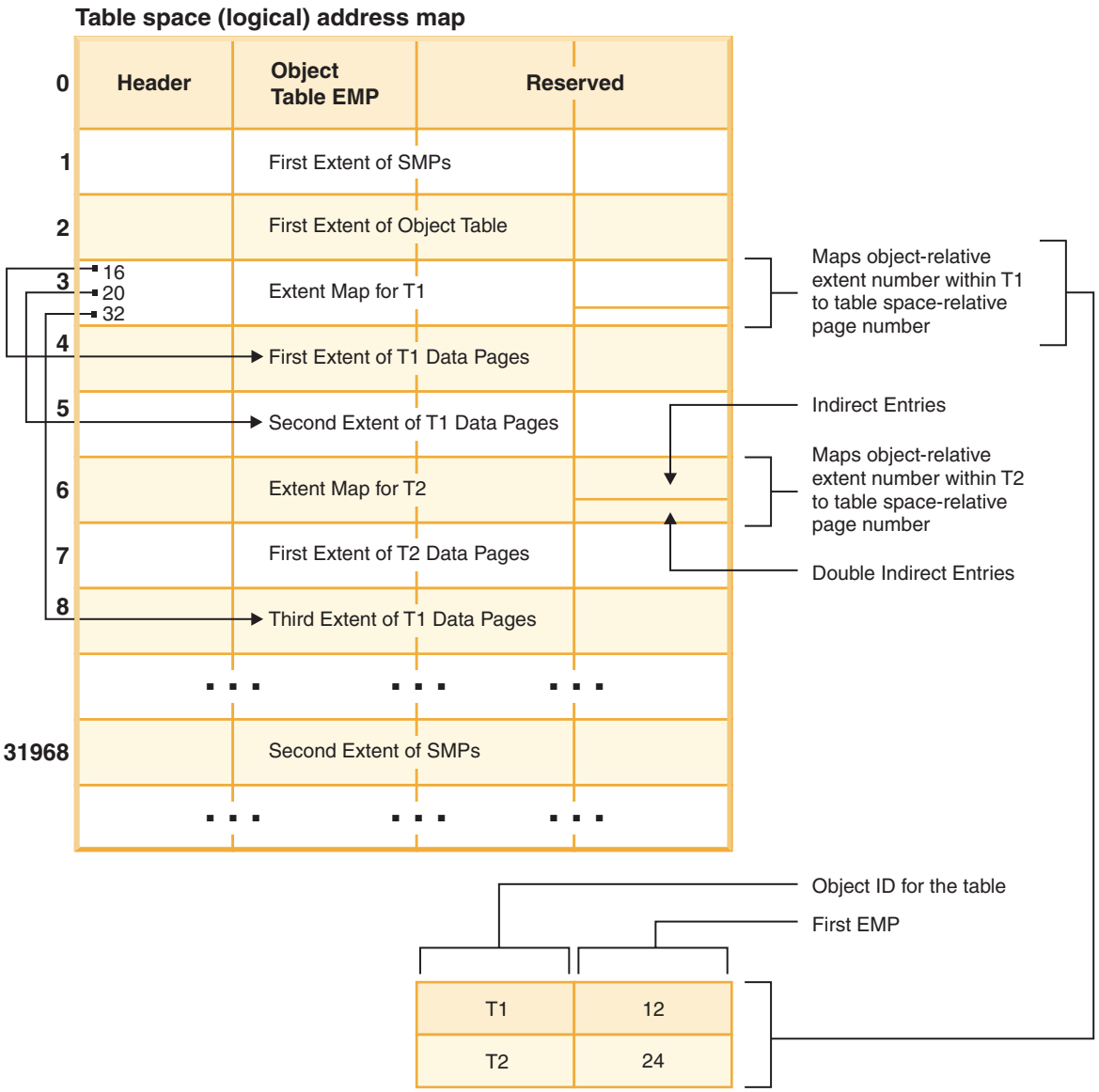


Figure 10. DMS table spaces

Within the table space address map, there are two types of map pages: extent map pages (EMP) and space map pages.

The object table is an internal relational table that maps an object identifier to the location of the first EMP extent in the table. This EMP extent, directly or indirectly, maps out all extents in the object. Each EMP contains an array of entries. Each entry maps an object-relative extent number to a table space-relative page number where the object extent is located. Direct EMP entries directly map object-relative addresses to table space-relative addresses. The last EMP page in the first EMP extent contains indirect entries. Indirect EMP entries map to EMP pages which then map to object pages. The last 16 entries in the last EMP page in the first EMP extent contain double-indirect entries.

The extents from the logical table space address map are striped in round-robin order across the containers associated with the table space.

Because space in containers is allocated by extent, pages that do not make up a full extent are not used. For example, if you have a 205-page container with an extent size of 10, one extent is used for the tag, 19 extents are available for data, and the five remaining pages are wasted.

If a DMS table space contains a single container, the conversion from logical page number to physical location on disk is a straightforward process where pages 0, 1, 2, are located in that same order on disk.

It is also a fairly straightforward process when there is more than one container and each of the containers is the same size. The first extent in the table space, containing pages 0 to (extent size - 1), is located in the first container, the second extent is located in the second container, and so on. After the last container, the process repeats starting back at the first container. This cyclical process keeps the data balanced.

For table spaces containing containers of different sizes, a simple approach that proceeds through each container in turn cannot be used as it will not take advantage of the extra space in the larger containers. This is where the table space map comes in - it dictates how extents are positioned within the table space, ensuring that all of the extents in the physical containers are available for use.

Note: In the following examples, the container sizes do not take the size of the container tag into account. The container sizes are very small, and are just used for illustration, they are not recommended container sizes. The examples show containers of different sizes within a table space, but you are advised to use containers of the same size.

Example 1:

There are 3 containers in a table space, each container contains 80 usable pages, and the extent size for the table space is 20. Each container therefore has 4 extents (80 / 20) for a total of 12 extents. These extents are located on disk as shown in Figure 11.

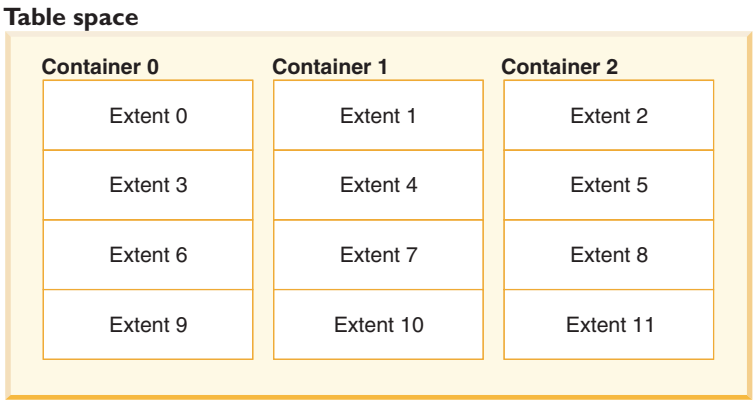


Figure 11. Table space with three containers and 12 extents

To see a table space map, take a table space snapshot using the snapshot monitor. In Example 1, where the three containers are of equal size, the table space map looks like this:

```
Range   Stripe Stripe Max      Max   Start   End    Adj.  Containers
Number  Set    Offset Extent   Page  Stripe  Stripe
   [0]    [0]       0     11    239       0       3   0   3 (0, 1, 2)
```

A *range* is the piece of the map in which a contiguous range of stripes all contain the same set of containers. In Example 1, all of the stripes (0 to 3) contain the same set of 3 containers (0, 1, and 2) and therefore this is considered a single range.

The headings in the table space map are Range Number, Stripe Set, Stripe Offset, Maximum extent number addressed by the range, Maximum page number addressed by the range, Start Stripe, End Stripe, Range adjustment, and Container list. These are explained in more detail for Example 2.

This table space can also be diagrammed as shown in Figure 12, in which each vertical line corresponds to a container, each horizontal line is called a *stripe*, and each cell number corresponds to an extent.

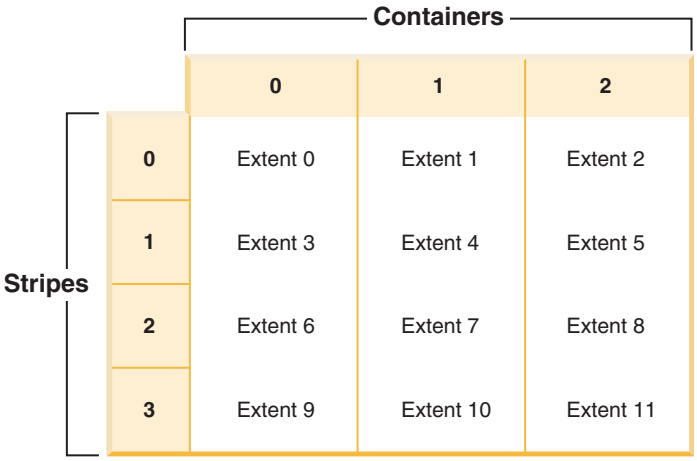


Figure 12. Table space with three containers and 12 extents, with stripes highlighted

Example 2:

There are two containers in the table space: the first is 100 pages in size, the second is 50 pages in size, and the extent size is 25. This means that the first container has four extents and the second container has two extents. The table space can be diagrammed as shown in Figure 13 on page 157.

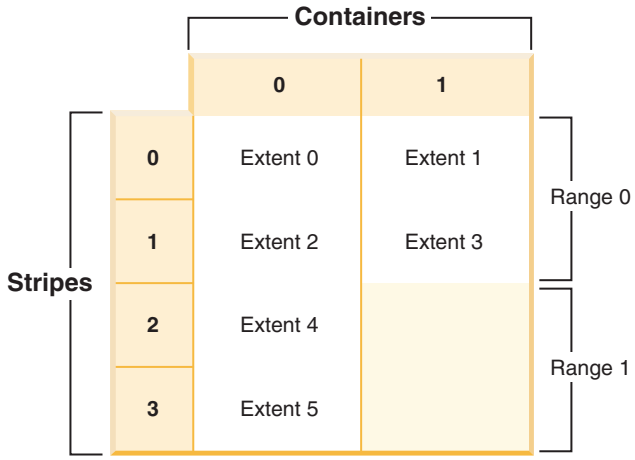


Figure 13. Table space with two containers, with ranges highlighted

Stripes 0 and 1 contain both of the containers (0 and 1) but stripes 2 and 3 only contain the first container (0). Each of these sets of stripes is a range. The table space map, as shown in a table space snapshot, looks like this:

```
Range   Stripe Stripe   Max      Max   Start   End    Adj.  Containers
Number  Set    Offset   Extent   Page  Stripe  Stripe
    [0]    [0]      0        3     99       0       1  0    2 (0, 1)
    [1]    [0]      0        5    149       2       3  0    1 (0)
```

There are four extents in the first range, and therefore the maximum extent number addressed in this range (Max Extent) is 3. Each extent has 25 pages and therefore there are 100 pages in the first range. Since page numbering also starts at 0, the maximum page number addressed in this range (Max Page) is 99. The first stripe (Start Stripe) in this range is 0 and the last stripe (End Stripe) in the range is stripe 1. There are two containers in this range and those are 0 and 1. The stripe offset is the first stripe in the stripe set, which in this case is 0 because there is only one stripe set. The range adjustment (Adj.) is an offset used when data is being rebalanced in a table space. (A rebalance might occur when space is added or dropped from a table space.) When a rebalance is not taking place, this is always 0.

There are two extents in the second range and because the maximum extent number addressed in the previous range is 3, the maximum extent number addressed in this range is 5. There are 50 pages (2 extents * 25 pages) in the second range and because the maximum page number addressed in the previous range is 99, the maximum page number addressed in this range is 149. This range starts at stripe 2 and ends at stripe 3.

Automatic re-sizing of DMS table spaces:

Enabling database-managed (DMS) table spaces that use file containers for automatic resizing allows the database manager to handle the full table space condition automatically by extending existing containers for you.

DMS table spaces are made up of file containers or raw device containers, and their sizes are set when the containers are assigned to the table space. The table space is considered to be full when all of the space within the containers has been used. However, unlike for SMS table spaces, you can add or extend containers manually, using the ALTER TABLESPACE statement, allowing more storage space to be given to the table space. DMS table spaces also have a feature called

auto-resize: as space is consumed in a DMS table space that can be automatically re-sized, the database manager increases the size of the table space by extending one or more file containers.

The auto-resize capability for DMS table spaces is related to, but different from capabilities of automatic storage table spaces. For more information see "Comparison of automatic storage, SMS, and DMS table spaces" on page 174.

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*.

Enabling and disabling the auto-resize feature

By default, the auto-resize feature is not enabled for a DMS table space. The following statement creates a DMS table space without enabling auto-resize:

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
```

To enable the auto-resize feature, specify the AUTORESIZE YES clause for the CREATE TABLESPACE statement:

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M) AUTORESIZE YES
```

You can also enable or disable the auto-resize feature after creating a DMS table space by using ALTER TABLESPACE statement with the AUTORESIZE clause:

```
ALTER TABLESPACE DMS1 AUTORESIZE YES
ALTER TABLESPACE DMS1 AUTORESIZE NO
```

Two other attributes, MAXSIZE and INCREASESIZE, are associated with auto-resize table spaces:

Maximum size (MAXSIZE)

The MAXSIZE clause of the CREATE TABLESPACE statement defines the maximum size for the table space. For example, the following statement creates a table space that can grow to 100 megabytes (per database partition if the database has multiple database partitions):

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
  AUTORESIZE YES MAXSIZE 100 M
```

The MAXSIZE NONE clause specifies that there is no maximum limit for the table space. The table space can grow until a file system limit or table space limit is reached (see "SQL and XML limits" in the *SQL Reference*). If you do not specify the MAXSIZE clause, there is no maximum limit when the auto-resize feature is enabled.

Use the ALTER TABLESPACE statement to change the value of MAXSIZE for a table space that has auto-resize already enabled, as shown in the following examples:

```
ALTER TABLESPACE DMS1 MAXSIZE 1 G
ALTER TABLESPACE DMS1 MAXSIZE NONE
```

If you specify a maximum size, the actual value that the database manager enforces might be slightly smaller than the value specified because the database manager attempts to keep container growth consistent.

Increase size (INCREASESIZE)

The INCREASESIZE clause of the CREATE TABLESPACE statement defines the amount of space used to increase the table space when there are no free extents within the table space but a request for one or more extents was made. You can specify the value as an explicit size or as a percentage, as shown in the following examples:

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
  AUTORESIZE YES INCREASESIZE 5 M

CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
  AUTORESIZE YES INCREASESIZE 50 PERCENT
```

A percentage value means that the amount by which to increase is calculated every time that the table space needs to grow; that is, growth is based on a percentage of the table space size at that point in time. For example, if the table space is 20 MB in size and the INCREASESIZE value is 50% , the table space grows by 10 MB the first time (to a size of 30 MB) and by 15 MB the next time.

If you do not specify the INCREASESIZE clause when you enable the auto-resize feature, the database manager determines an appropriate value to use, which might change over the life of the table space. As with AUTORESIZE and MAXSIZE, you can change the value of INCREASESIZE using the ALTER TABLESPACE statement.

If you specify a size increase, the actual value that the database manager will use might be slightly different than the value that you provide. This adjustment in the value used is done to keep growth consistent across the containers in the table space.

Restrictions for using AUTORESIZE with DMS table spaces

- You cannot use this feature for table spaces that use raw device containers, and you cannot add raw device containers to a table space that can be automatically resized. Attempting these operations results in errors (SQL0109N). If you need to add raw device containers, you must disable the auto-resize feature first.
- If you disable the auto-resize feature, the values that are associated with INCREASESIZE and MAXSIZE are not retained if you subsequently enable this feature.
- A redirected restore operation cannot change the container definitions to include a raw device container. Attempting this kind of operation results in an error (SQL0109N).
- In addition to limiting how the database manager automatically increases a table space, the maximum size also limits the extent to which you can manually increase a table space. If you perform an operation that adds space to a table space, the resulting size must be less than or equal to the maximum size. You can add space by using the ADD, EXTEND, RESIZE, or BEGIN NEW STRIPE SET clause of the ALTER TABLESPACE statement.

How table spaces are extended

When AUTORESIZE is enabled, the database manager attempts to increase the size of the table space when all of the existing space has been used and a request for more space is made. The database manager determines which of the containers can be extended in the table space so that a rebalancing of the data in the table space does not occur. The database manager extends only those containers that exist within the last range of the table space map (the map describes the storage layout for the table space - see "Table space maps for database-managed table spaces" on page 153 for more information) and extends them by an equal amount.

For example, consider the following statement:

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE
  USING (FILE 'C:\TS1CONT' 1000, FILE 'D:\TS1CONT' 1000,
         FILE 'E:\TS1CONT' 2000, FILE 'F:\TS1CONT' 2000)
  EXTENTSIZE 4
  AUTORESIZE YES
```

Keeping in mind that the database manager uses a small portion (one extent) of each container for metadata, following is the table space map that is created for the table space based on the CREATE TABLESPACE statement. (The table space map is part of the output from a table space snapshot.)

```
Table space map:

 Range  Stripe Stripe  Max         Max  Start  End    Adj.   Containers
 Number Set    Offset  Extent      Page Stripe End    Stripe
 [   0] [   0]      0     995      3983      0    248   0    4 (0,1,2,3)
 [   1] [   0]      0    1495      5983    249    498   0    2 (2,3)
```

The table space map shows that the containers with an identifier of 2 or 3 (`E:\TS1CONT` and `F:\TS1CONT`) are the only containers in the last range of the map. Therefore, when the database manager automatically extends the containers in this table space, it extends only those two containers.

Note: If you create a table space with all of the containers having the same size, there is only one range in the map. In such a case, the database manager extends each of the containers. To prevent restricting extensions to only a subset of the containers, create a table space with containers of equal size.

As discussed previously, you can specify a limit on the maximum size of the table space, or you can specify a value of NONE, which does not limit growth. If you specify NONE or no limit, the upper limit is defined by the file system limit or by the table space limit; the database manager does not attempt to increase the table space size past the upper limit. However, before that limit is reached, an attempt to increase a container might fail due to a full file system. In this case, the database manager does not increase the table space size any further and returns an out-of-space condition to the application. There are two ways to resolve this situation:

- Increase the amount of space available on the file system that is full.
- Perform container operations on the table space such that the container in question is no longer in the last range of the table space map. The easiest way to do this is to add a new stripe set to the table space with a new set of containers, and the best practice is to ensure that the containers are all the same size. You can add new stripe sets by using the ALTER TABLESPACE statement with the BEGIN NEW STRIPE SET clause. By adding a new stripe set, a new range is added to the table space map. With a new range, the containers that the

database manager automatically attempts to extend are within this new stripe set, and the older containers remain unchanged.

Note: When a user-initiated container operation is pending or a subsequent rebalance is in progress, the auto-resize feature is disabled until the operation is committed or the rebalance is complete.

For example, for DMS table spaces, suppose that a table space has three containers that are the same size and that each resides on its own file system. As work is done on the table space, the database manager automatically extends these three containers. Eventually, one of the file systems becomes full, and the corresponding container can no longer grow. If more free space cannot be made available on the file system, you must perform container operations on the table space such that the container in question is no longer in the last range of the table space map. In this case, you could add a new stripe set specifying two containers (one on each of the file systems that still has space), or you could specify more containers (again, making sure that each container being added is the same size and that there is sufficient room for growth on each of the file systems being used). When the database manager attempts to increase the size of the table space, it now attempts to extend the containers in this new stripe set instead of attempting to extend the older containers.

Monitoring

Information about automatic resizing for DMS table spaces is displayed as part of the table space monitor snapshot output. The increase size and maximum size values are included in the output, as shown in the following sample:

```
Auto-resize enabled                     = Yes or No
Current tablespace size (bytes)         = ###
Maximum tablespace size (bytes)         = ### or NONE
Increase size (bytes)                   = ###
Increase size (percent)                 = ###
Time of last successful resize          = DD/MM/YYYY HH:MM:SS.SSSSSS
Last resize attempt failed              = Yes or No
```

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 14 on page 163 illustrates this process.

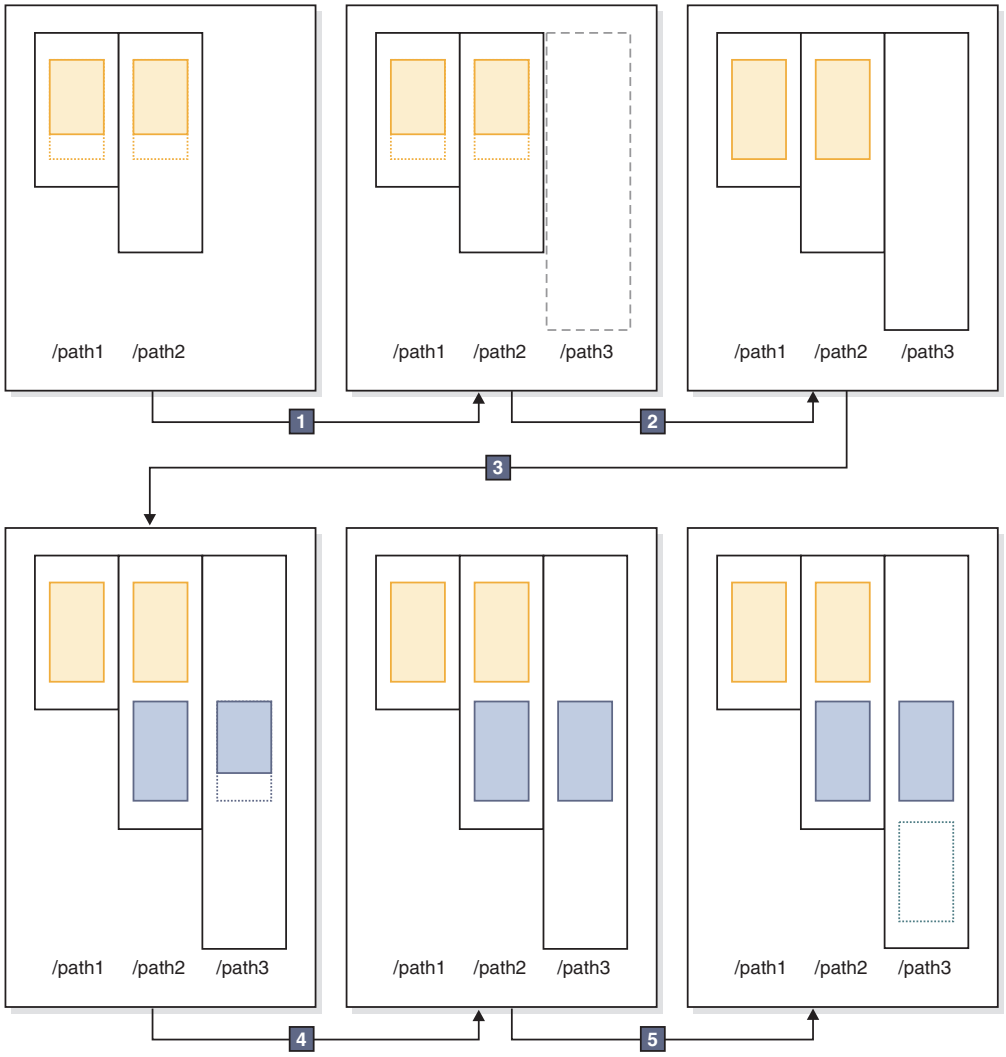


Figure 14. How automatic storage adds containers as needed

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 15 on page 164.

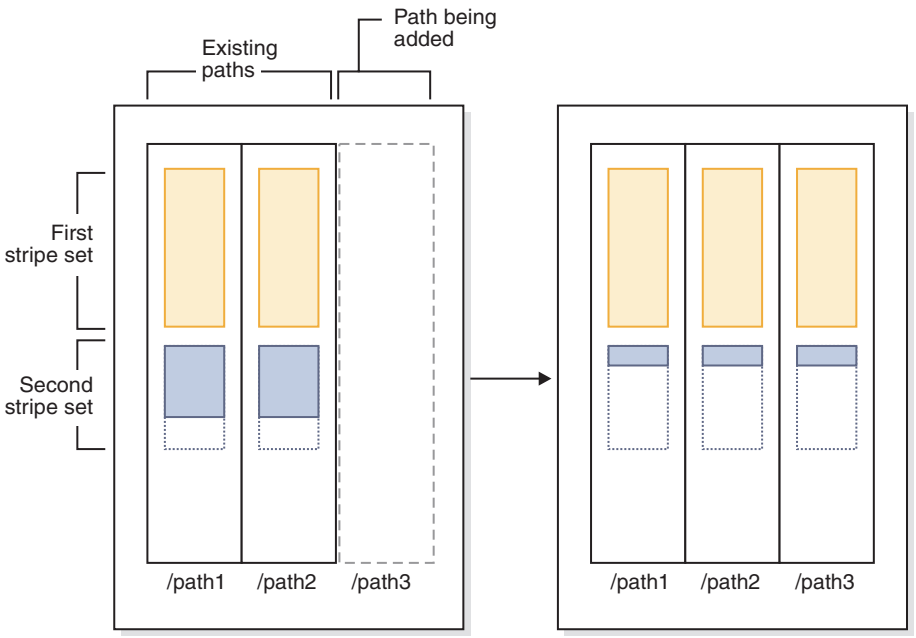


Figure 15. Results of adding new storage and rebalancing the table space

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.

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 16 on page 168 shows a series of allocated extents in a table space.

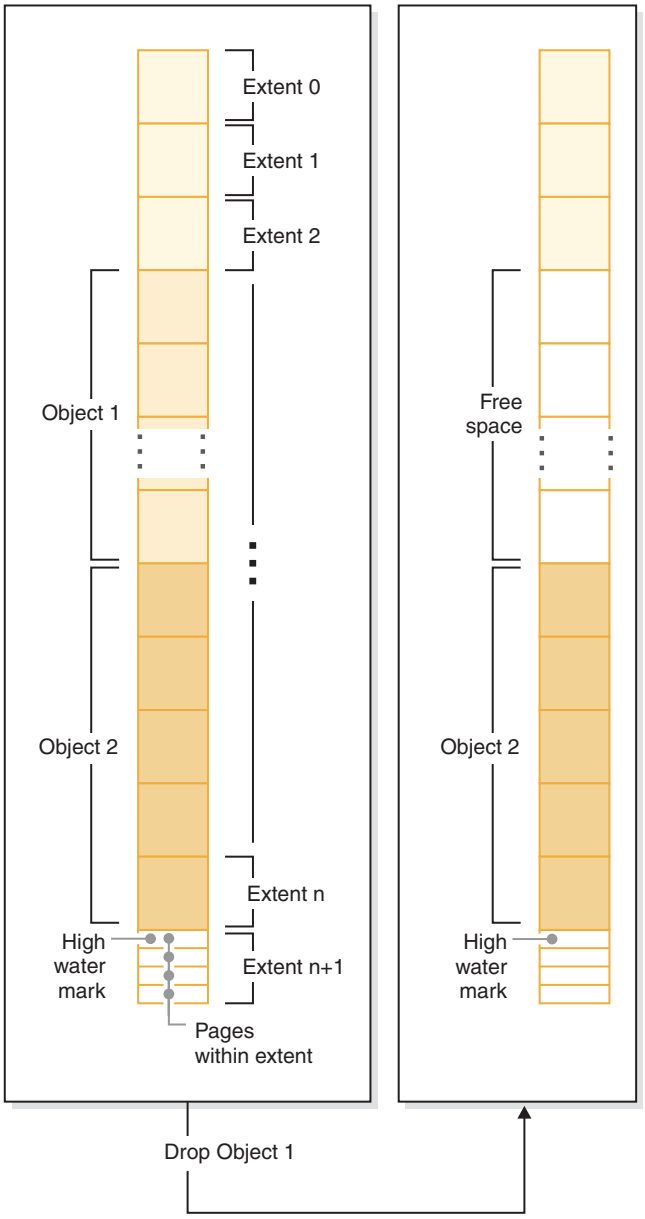


Figure 16. High water mark

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 17 shows a high-level view of how reclaimable storage works.

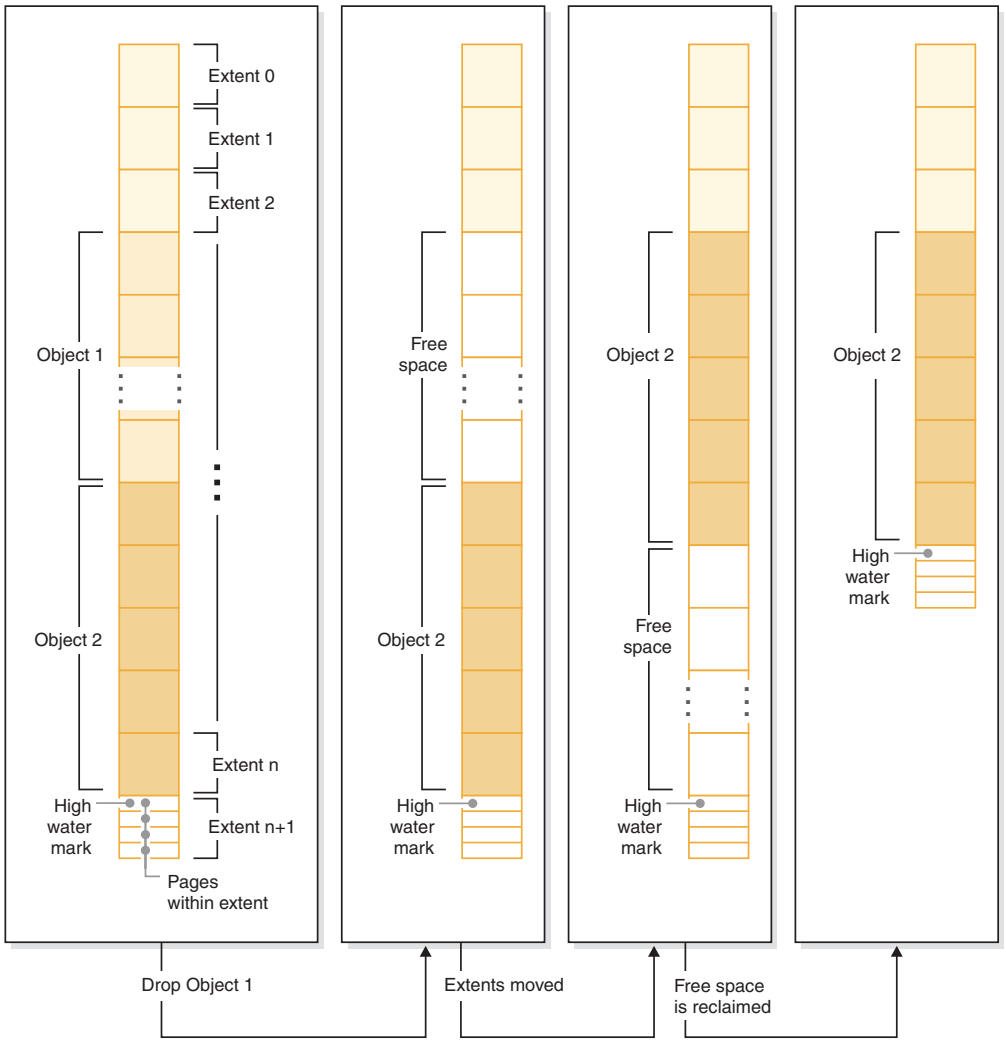


Figure 17. How reclaimable storage works. When reclaimable storage is enabled for a table space, the in-use extents can be moved to occupy unused extents lower in the table space.

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 18 on page 173

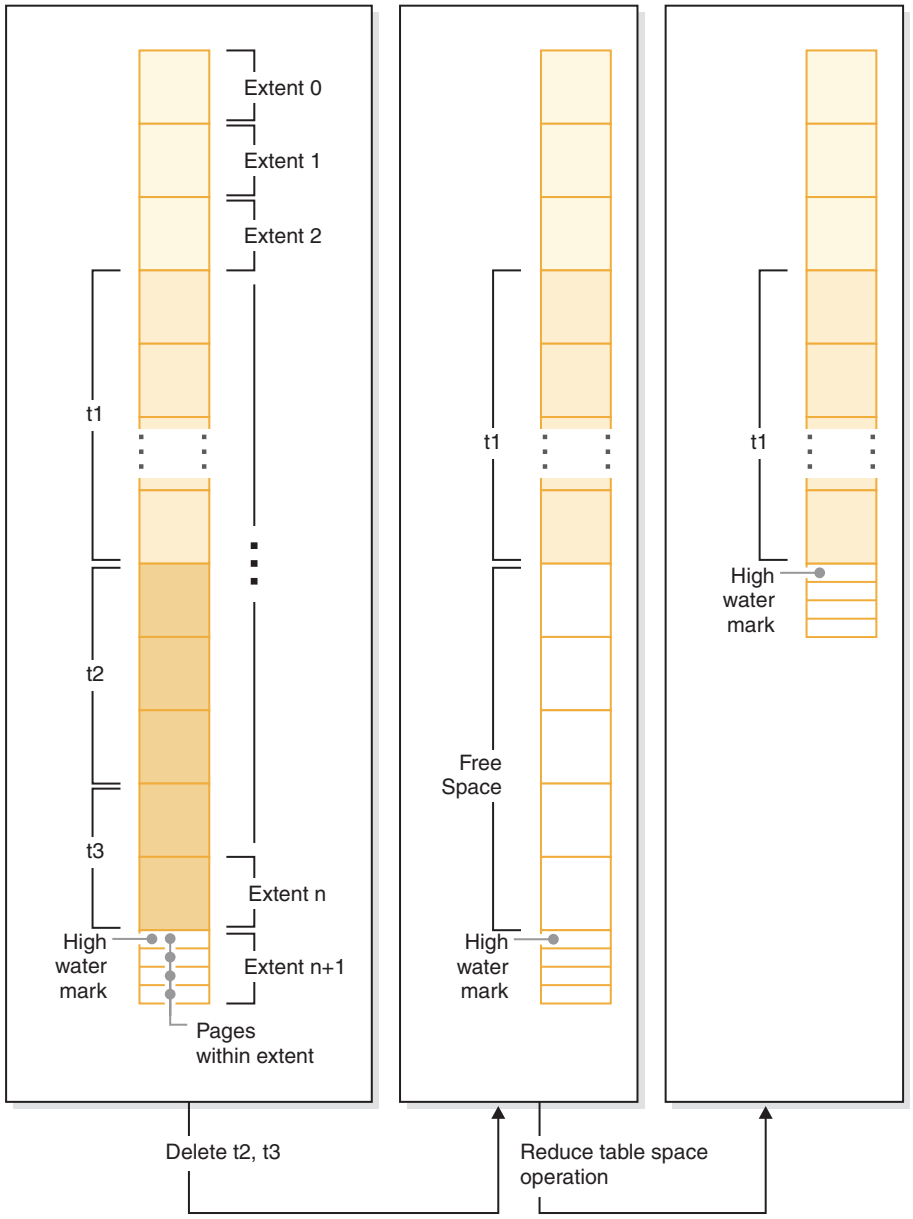


Figure 18. Lowering the high water mark without needing to move extents.

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)
```

Comparison of automatic storage, SMS, and DMS table spaces

Automatic storage, SMS, and DMS table spaces offer different capabilities that can be advantageous in different circumstances.

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*

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*.

Table 12. Comparison of SMS, DMS and automatic storage table spaces

	Automatic storage table spaces	**SMS table spaces**	**DMS table spaces**
How they are created	Created using the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or by omitting the MANAGED BY clause entirely. If the automatic storage was enabled when the database was created, the default for any table space you create is to create it as an automatic storage table space unless you specify otherwise.	Created using the MANAGED BY SYSTEM clause of the CREATE TABLESPACE statement	Created using the MANAGED BY DATABASE clause of the CREATE TABLESPACE statement
Initial container definition and location	You do not provide a list of containers when creating an automatic storage table space. Instead, the database manager automatically creates containers on all of the storage paths associated with the database. Data is striped evenly across all containers so that the storage paths are used equally.	Requires that containers be defined as a directory name.	• Requires that containers be defined as files or devices. • Must specify the initial size for each container.
Initial allocation of space	• For nontemporary automatic storage table spaces: – Space is allocated when the table space is created – You can specify the initial size for table space • For temporary automatic storage table spaces, space is allocated as needed.	Done as needed. Because the file system controls the allocation of storage, there is less likelihood that pages will be contiguous, which could have an impact on the performance of some types of queries.	Done when table space created. • Extents are more likely to be contiguous than they would be with SMS table spaces. • Pages within extents are always contiguous for device containers.

Table 12. Comparison of SMS, DMS and automatic storage table spaces (continued)

	Automatic storage table spaces	**SMS table spaces**	**DMS table spaces**
Changes to table space containers	• Containers can dropped or reduced if the table space size is reduced. • Table space can be rebalanced to distribute data evenly across containers when new storage is added to or dropped from the database.	No changes once created, other than to add containers for new data partitions as they are added.	• Containers can be extended or added. A rebalance of the table space data will occur if the new space is added below the high water mark for the table space. • Containers can be reduced or dropped. A rebalance will occur if there is data in the space being dropped
Handling of demands for increased storage	• Containers are extended automatically up to constraints imposed by file system. • If storage paths are added to the database, containers are extended or created automatically.	Containers will grow until they reach the capacity imposed by the file system. The table space is considered to be full when any one container reaches its maximum capacity.	Containers can be extended beyond the initially-allocated size manually or automatically (if auto-resize is enabled) up to constraints imposed by file system.
Ability to place different types of objects in different table spaces	Tables, storage for related large objects (LOBs) and indexes can each reside in separate table spaces.	For partitioned tables only, indexes and index partitions can reside in a table space separate from the one containing table data.	Tables, storage for related large objects (LOBs) and indexes can each reside in separate table spaces.
Ongoing maintenance requirements	• Reducing size of table space • Lowering high water mark • Rebalancing	None	• Adding or extending containers • Dropping or reducing containers • Lowering high water mark • Rebalancing
Use of restore to redefine containers	You cannot use a redirected restore operation to redefine the containers associated with the table space because the database manager manages space.	You can use a redirected restore operation to redefine the containers associated with the table space	You can use a redirected restore operation to redefine the containers associated with the table space
Performance	Similar to DMS	Generally slower than DMS and automatic storage, especially for larger tables.	Generally superior to SMS

Automatic storage table spaces are the easiest table spaces to set up and maintain, and are recommended for most applications. They are particularly beneficial when:

- You have larger tables or tables that are likely to grow quickly
- You do not want to have to make regular decisions about how to manage container growth.
- You want to be able to store different types of related objects (for example, tables, LOBs, indexes) in different table spaces to enhance performance.

SMS and DMS workload considerations:

The primary type of workload being managed by the database manager in your environment can affect your choice of what table space type to use, and what page size to specify.

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*

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*.

An online transaction processing (OLTP) workload is characterized by transactions that need random access to data, often involve frequent insert or update activity and queries which usually return small sets of data. Given that the access is random, and involves one or a few pages, prefetching is less likely to occur.

DMS table spaces using device containers perform best in this situation. DMS table spaces with file containers are also reasonable choices for OLTP workloads if maximum performance is not required. Note that using DMS table spaces with file containers, where FILE SYSTEM CACHING is turned off, can perform at a level comparable to DMS raw table space containers. With little or no sequential I/O expected, the settings for the EXTENTSIZE and the PREFETCHSIZE parameters on the CREATE TABLESPACE statement are not important for I/O efficiency. However, setting a sufficient number of page cleaners, using the *chngpgs_thresh* configuration parameter, is important.

A query workload is characterized by transactions that need sequential or partially sequential access to data, which usually return large sets of data. A DMS table space using multiple device containers (where each container is on a separate disk) offers the greatest potential for efficient parallel prefetching. The value of the PREFETCHSIZE parameter on the CREATE TABLESPACE statement should be set to the value of the EXTENTSIZE parameter, multiplied by the number of device containers. Alternatively, you can specify a prefetch size of -1 and the database manager automatically chooses an appropriate prefetch size. This allows the database manager to prefetch from all containers in parallel. If the number of containers changes, or there is a need to make prefetching more or less aggressive, the PREFETCHSIZE value can be changed accordingly by using the ALTER TABLESPACE statement.

A reasonable alternative for a query workload is to use files, if the file system has its own prefetching. The files can be either of DMS type using file containers, or of SMS type. Note that if you use SMS, you must have the directory containers map to separate physical disks to achieve I/O parallelism.

Your goal for a mixed workload is to make single I/O requests as efficient as possible for OLTP workloads, and to maximize the efficiency of parallel I/O for query workloads.

The considerations for determining the page size for a table space are as follows:

- For OLTP applications that perform random row read and write operations, a smaller page size is usually preferable because it does not waste buffer pool space with unwanted rows.
- For decision-support system (DSS) applications that access large numbers of consecutive rows at a time, a larger page size is usually better because it reduces the number of I/O requests that are required to read a specific number of rows.
- Larger page sizes might allow you to reduce the number of levels in the index.
- Larger pages support rows of greater length.
- On default 4 KB pages, tables are restricted to 500 columns, whereas the 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.

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*.

SMS and DMS device considerations:

There are a few options to consider when choosing to use file system files versus devices for table space containers: the buffering of data and whether to use LOB or LONG data.

- **Buffering of data**

  Table data read from disk is usually available in the database buffer pool. In some cases, a data page might be freed from the buffer pool before the application has actually used the page, particularly if the buffer pool space is required for other data pages. For table spaces that use system managed space (SMS) or database managed space (DMS) file containers, file system caching can eliminate I/O that would otherwise have been required.

  Table spaces using database managed space (DMS) device containers do not use the file system or its cache. As a result, you might increase the size of the database buffer pool and reduce the size of the file system cache to offset the fact DMS table spaces that use device containers do not use double buffering.

  If system-level monitoring tools show that I/O is higher for a DMS table space using device containers compared to the equivalent SMS table space, this difference might be because of double buffering.

  Important: User table spaces that use System Managed Space (SMS) are deprecated and might be removed in a future release. Use Database Managed Spaces (DMS) or Automatic Storage table spaces (AMS) instead.
- **Using LOB or LONG data**

  When an application retrieves either LOB or LONG data, the database manager does not cache the data in its buffers. Each time an application needs one of these pages, the database manager must retrieve it from disk. However, if LOB or LONG data is stored in DMS file containers, file system caching might provide buffering and, as a result, better performance.

  Because system catalogs contain some LOB columns, you should keep them in DMS-file table spaces.

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.

A database must have at least one *system* temporary table space with the same page size as the catalog table space. By default, one system temporary table space called TEMPSPACE1 is created at database creation time. IBMTEMPGROUP is the default database partition group for this table space. The page size for TEMPSPACE1 is whatever was specified when the database itself was created (by default, 4 kilobytes).

User temporary table spaces hold temporary data from tables created with a DECLARE GLOBAL TEMPORARY TABLE or CREATE GLOBAL TEMPORARY TABLE statement. User temporary table spaces are not created by default at the time of database creation. They also hold instantiated versions of created temporary tables.

It is recommended that you define a single temporary table space with a page size equal to the page size used in the majority of your user table spaces. This should be suitable for typical environments and workloads. However, it can be advantageous to experiment with different temporary table space configurations and workloads. The following points should be considered:

- Temporary tables are in most cases accessed in batches and sequentially. That is, a batch of rows are inserted, or a batch of sequential rows are fetched. Therefore, a larger page size typically results in better performance, because fewer logical and physical page requests are required to read a given amount of data.
- When reorganizing a table using a temporary table space, the page size of the temporary table space must match that of the table. For this reason, you should ensure that there are temporary table spaces defined for each different page size used by existing tables that you might reorganize using a temporary table space.

  You can also reorganize without a temporary table space by reorganizing the table directly in the same table space. This type of reorganization requires that there be extra space in the table space(s) of the table for the reorganization process.
- When using SMS system temporary table spaces, you might want to consider using the registry variable DB2_SMS_TRUNC_TMPTABLE_THRESH. When dropped, files created for the system temporary tables are truncated to a size of 0. The DB2_SMS_TRUNC_TMPTABLE_THRESH can be used to avoid visiting the file systems and potentially leave the files at a non-zero size to avoid the performance cost of repeated extensions and truncations of the files.
- In general, when temporary table spaces of different page sizes exist, the optimizer will choose the temporary table space whose buffer pool can hold the most number of rows (in most cases that means the largest buffer pool). In such cases, it is often wise to assign an ample buffer pool to one of the temporary table spaces, and leave any others with a smaller buffer pool. Such a buffer pool assignment will help ensure efficient utilization of main memory. For example, if your catalog table space uses 4 KB pages, and the remaining table spaces use 8 KB pages, the best temporary table space configuration might be a single 8 KB temporary table space with a large buffer pool, and a single 4 KB table space with a small buffer pool.
- There is generally no advantage to defining more than one temporary table space of any single page size.

Automatic storage temporary table spaces, like regular and large automatic storage table spaces, are associated with storage groups. However, automatic storage temporary table spaces cannot change their storage group association. If a rebalance operation is attempted on an automatic storage temporary table space, SQL0109N is returned. To associate a temporary table space with a storage group, you can drop the temporary table space and re-create it using a different storage group. If you add storage paths to a storage group, temporary table spaces do not take advantage of the new paths until the next database activation.

Considerations when choosing table spaces for your tables

When determining how to map tables to tables spaces, you should consider the distribution of your tables, the amount and type of data in the table, and administrative issues.

The distribution of your tables

At a minimum, you should ensure that the table space you choose is in a database partition group with the distribution you want.

The amount of data in the table

If you plan to store many small tables in a table space, consider using SMS for that table space. The DMS advantages with I/O and space management efficiency are not as important with small tables. The SMS advantages, and only when needed, are more attractive with smaller tables. If one of your tables is larger, or you need faster access to the data in the tables, a DMS table space with a small extent size should be considered.

You might want to use a separate table space for each very large table, and group all small tables together in a single table space. This separation also allows you to select an appropriate extent size based on the table space usage.

Important: User table spaces that use System Managed Space (SMS) are deprecated and might be removed in a future release. Use Database Managed Spaces (DMS) or Automatic Storage table spaces (AMS) instead.

The type of data in the table

You might, for example, have tables containing historical data that is used infrequently; the end-user might be willing to accept a longer response time for queries executed against this data. In this situation, you could use a different table space for the historical tables, and assign this table space to less expensive physical devices that have slower access rates.

Alternatively, you might be able to identify some essential tables for which the data has to be readily available and for which you require fast response time. You might want to put these tables into a table space assigned to a fast physical device that can help support these important data requirements.

Using DMS table spaces, you can also distribute your table data across four different table spaces: one for index data; one for large object (LOB) and long field (LF) data; one for regular table data, and one for XML data. This allows you to choose the table space characteristics and the physical devices supporting those table spaces to best suit the data. For example, you could put your index data on the fastest devices you have available, and as a result, obtain significant performance improvements. If you split a table across DMS table spaces, you should consider backing up and

restoring those table spaces together if rollforward recovery is enabled. SMS table spaces do not support this type of data distribution across table spaces.

Administrative issues

Some administrative functions can be performed at the table space level instead of the database or table level. For example, taking a backup of a table space instead of a database can help you make better use of your time and resources. It allows you to frequently back up table spaces with large volumes of changes, while only occasionally backing up tables spaces with very low volumes of changes.

You can restore a database or a table space. If unrelated tables do not share table spaces, you have the option to restore a smaller portion of your database and reduce costs.

A good approach is to group related tables in a set of table spaces. These tables could be related through referential constraints, or through other defined business constraints.

If you need to drop and redefine a particular table often, you might want to define the table in its own table space, because it is more efficient to drop a DMS table space than it is to drop a table.

Table spaces without file system caching

The recommended method of enabling or disabling non-buffered I/O on UNIX, Linux, and Windows is at the table space level.

This allows you to enable or disable non-buffered I/O on specific table spaces while avoiding any dependency on the physical layout of the database. It also allows the database manager to determine which I/O is best suited for each file, buffered or non-buffered.

The NO FILE SYSTEM CACHING clause is used to enable non-buffered I/O, thus disabling file caching for a particular table space. Once enabled, based on platform, the database manager automatically determines which of the Direct I/O (DIO) or Concurrent I/O (CIO) is to be used. Given the performance improvement in CIO, the database manager uses it whenever it is supported; there is no user interface to specify which one is to be used.

In order to obtain the maximum benefits of non-buffered I/O, it might be necessary to increase the size of buffer pools. However, if the self-tuning memory manager is enabled and the buffer pool size is set to AUTOMATIC, the database manager will self-tune the buffer pool size for optimal performance. Note that this feature is not available before Version 9.

To disable or enable file system caching, specify the NO FILE SYSTEM CACHING or the FILE SYSTEM CACHING clause in the CREATE TABLESPACE or ALTER TABLESPACE statement. The default setting is used if neither clause is specified. In the case of ALTER TABLESPACE, existing connections to the database must be terminated before the new caching policy takes effect.

Note: If an attribute is altered from the default to either FILE SYSTEM CACHING or NO FILE SYSTEM CACHING, there is no mechanism to change it back to the default.

This method of enabling and disabling file system caching provides control of the I/O mode, buffered or non-buffered, at the table space level.

To determine whether file system caching is enabled, query the value of the **fs_caching** monitor element for the table space in the MON_GET_TABLESPACE table.

Alternate methods to enable/disable non-buffered I/O on UNIX, Linux, and Windows

Some UNIX platforms support the disabling of file system caching at a file system level by using the MOUNT option. Consult your operating system documentation for more information. However, it is important to understand the difference between disabling file system caching at the table space level and at the file system level. At the table space level, the database manager controls which files are to be opened with and without file system caching. At the file system level, every file residing on that particular file system will be opened without file system caching. Some platforms such as AIX have certain requirements before you can use this feature, such as serialization of read and write access. although the database manager adheres to these requirements, if the target file system contains files not from the database manager, before enabling this feature, consult your operating system documentation for any requirements.

Note: The now-deprecated registry variable **DB2_DIRECT_IO**, introduced in Version 8.1 FixPak 4, enables no file system caching for all SMS containers except for long field data, large object data, and temporary table spaces on AIX JFS2. Setting this registry variable in Version 9.1 or later is equivalent to altering all table spaces, SMS and DMS, with the NO FILE SYSTEM CACHING clause. However, using **DB2_DIRECT_IO** is not recommended, and this variable will be removed in a later release. Instead, you should enable NO FILE SYSTEM CACHING at the table space level.

Alternate methods to enable/disable non-buffered I/O on Windows

In previous releases, the performance registry variable **DB2NTNOCACHE** could be used to disable file system caching for all DB2 files in order to make more memory available to the database so that the buffer pool or sort heap can be increased. The difference between **DB2NTNOCACHE** and using the NO FILE SYSTEM CACHING clause is the ability to disable caching for selective table spaces. Starting in Version 9.5, since the NO FILE SYSTEM CACHING is used as the default, unless FILE SYSTEM CACHING is specified explicitly, there is no need to set this registry variable to disable file system caching across the entire instance if the instance includes only newly created table spaces.

Performance considerations

Non-buffered I/O is essentially used for performance improvements. In some cases, however, performance degradation might be due to, but is not limited to, a combination of a small buffer pool size and a small file system cache. Suggestions for improving performance include:

- If self-tuning memory manager is not enabled, enable it and set the buffer pool size to automatic using `ALTER BUFFERPOOL name SIZE AUTOMATIC`. This allows the database manager to self-tune the buffer pool size.
- If self-tuning memory manager is not to be enabled, increase the buffer pool size in increments of 10 or 20 percent until performance is improved.

- If self-tuning memory manager is not to be enabled, alter the table space to use "FILE SYSTEM CACHING". This essentially disables the non-buffered I/O and reverts back to buffered I/O for container access.

Performance tuning should be tested in a controlled environment before implementing it on the production system.

When choosing to use file system files versus devices for table space containers, you should consider file system caching, which is performed as follows:

- For DMS file containers (and all SMS containers), the operating system might cache pages in the file system cache (unless the table space is defined with NO FILESYSTEM CACHING).
- For DMS device container table spaces, the operating system does not cache pages in the file system cache.

Table space containers use concurrent I/O or direct I/O by default

The default I/O mechanism for created table space containers on most AIX, Linux, Solaris, and Windows operating systems is CIO/DIO (concurrent I/O or Direct I/O). This default provides an increase of throughput over buffered I/O on heavy transaction processing workloads and rollbacks.

The FILE SYSTEM CACHING or NO FILE SYSTEM CACHING attribute specifies whether I/O operations are to be cached at the file system level:

- FILE SYSTEM CACHING specifies that all I/O operations in the target table space are to be cached at the file system level.
- NO FILE SYSTEM CACHING specifies that all I/O operations are to bypass the file system-level cache.

If large object (LOB) data is inlined, then it is accessed as regular data and uses the I/O method (buffered or non-buffered) specified for the table space FILE SYSTEM CACHING attribute.

If large object (LOB) data is not inlined, then the following statements apply:

- For SMS table spaces, non-buffered I/O access is not requested for long field (LF) data and large object (LOB) data even when the NO FILE SYSTEM CACHING table space attribute is set. Buffering occurs in the file system cache, subject to operating system configuration and behavior, and potentially improves performance.
- For DMS table spaces, DB2 does not distinguish between different data types when performing I/O. Buffering of LF or LOB data does not occur unless the table space is configured with FILE SYSTEM CACHING enabled. If buffering of LF or LOB data in DMS tables spaces is wanted for performance reasons, then you can place this data in a separate DMS table space and explicitly enable FILE SYSTEM CACHING.

The following interfaces contain the FILE SYSTEM CACHING attribute:

- CREATE TABLESPACE statement
- **CREATE DATABASE** command
- sqlecrea() API (using the *sqlfscaching* field of the SQLETSDESC structure)

When this attribute is not specified on the CREATE TABLESPACE statement, or on the **CREATE DATABASE** command, the database manager processes the request using the default behavior based on the platform and file system type. See "File system

caching configurations" for the exact behavior. For the sqlecrea() API, a value of `0x2` for the *sqlfscaching* field, instructs the database manager to use the default setting.

Note that the following tools currently interpret the value for FILE SYSTEM CACHING attribute:

- **GET SNAPSHOT FOR TABLESPACES** command
- **db2pd -tablespaces** command
- `db2look -d dbname -l` command

For **db2look**, if the FILE SYSTEM CACHING attribute is not specified, the output does not contain this attribute.

Example

Suppose that the database and all related table space containers reside on an AIX JFS file system and the following statement was issued:

```
DB2 CREATE TABLESPACE JFS2
```

If the attribute was not specified, the database manager uses NO FILE SYSTEM CACHING.

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

Table 13. Supported configurations for table spaces without file system caching (continued)

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
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
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.

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 145 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: (1) SYSCATSPACE for the system catalog tables, (2) TEMPSPACE1 for system temporary tables created during database processing, and (3) USERSPACE1 for user-defined tables and indexes. You can also create additional user table spaces at the same time.

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.

Attaching DMS direct disk access devices

When working with containers to store data, the database manager supports direct disk access (raw I/O).

About this task

This type of support allows you to attach a direct disk access (raw) device to any DB2 database system.

You must know the device or file names of the containers you are going to reference when creating your table spaces. You must know the amount of space associated with each device or file name that is to be allocated to the table space. You will need the correct permissions to read and write to the container.

The physical and logical methods for identifying direct disk access differs based on operating system:

- On the Windows operating systems:

  To specify a physical hard drive, use the following syntax:

  ```
  \\.\PhysicalDriveN
  ```

  where N represents one of the physical drives in the system. In this case, N could be replaced by 0, 1, 2, or any other positive integer:

  ```
  \\.\PhysicalDrive5
  ```

  To specify a logical drive, that is, an unformatted database partition, use the following syntax:

  ```
  \\.\N:
  ```

  where N: represents a logical drive letter in the system. For example, N: could be replaced by E: or any other drive letter. To overcome the limitation imposed by using a letter to identify the drive, you can use a globally unique identifier (GUID) with the logical drive.

  For Windows, there is a new method for specifying DMS raw table space containers. Volumes (that is, basic disk database partitions or dynamic volumes) are assigned a globally unique identifier (GUID) when they are created. The GUID can be used as a device identifier when specifying the containers in a table space definition. The GUIDs are unique across systems which means that in a multi-partition database, GUIDs are different for each database partition even if the disk partition definitions are the same.

  A tool called *db2listvolumes.exe* is available (only on Windows operating systems) to make it easy to display the GUIDs for all the disk volumes defined on a Windows system. This tool creates two files in the current directory where the tool is run. One file, called `volumes.xml`, contains information about each disk volume encoded in XML for easy viewing on any XML-enabled browser. The second file, called `tablespace.ddl`, contains the required syntax for specifying table space containers. This file must be updated to specify the remaining

information needed for a table space definition. The **db2listvolumes** command does not require any command line arguments.

- On Linux and UNIX platforms, a logical volume can appear to users and applications as a single, contiguous, and extensible disk volume. Although it appears this way, it can reside on noncontiguous physical database partitions or even on more than one physical volume. The logical volume must also be contained within a single volume group. There is a limit of 256 logical volumes per volume group. There is a limit of 32 physical volumes per volume group. You can create additional logical volumes using the **mklv** command. This command allows you to specify the name of the logical volume and to define its characteristics, including the number and location of logical partitions to allocate for it.

  After you create a logical volume, you can change its name and characteristics with the **chlv** command, and you can increase the number of logical partitions allocated to it with the **extendlv** command. The default maximum size for a logical volume at creation is 512 logical partitions, unless specified to be larger. The **chlv** command is used to override this limitation.

  Within AIX, the set of operating system commands, library subroutines, and other tools that allow you to establish and control logical volume storage is called the Logical Volume Manager (LVM). The LVM controls disk resources by mapping data between a simpler and flexible logical view of storage space and the actual physical disks.

  For more information about the **mklv** and other logical volume commands, and the LVM, refer to *AIX 5L Version 5.2 System Management Concepts: Operating System and Devices*.

Configuring and setting up DMS direct disk access (Linux)

When working with containers to store data, the database manager supports direct disk (raw) access using the block device interface (that is, raw I/O).

Before you begin

Before setting up raw I/O on Linux, one or more free IDE or SCSI disk database partitions are required. In order to reference the disk partition when creating the table space, you must know the name of the disk partition and the amount of space associated with the disk partition that is to be allocated to the table space.

About this task

Before Version 9, direct disk access using a raw controller utility on Linux was used. This method is now deprecated, and its use is discouraged. The database manager still allows you to use this method if the Linux operating system still supports it, however, a message is written in the **db2diag** log files to indicates that its use is deprecated.

The prior method required you to "bind" a disk partition to a raw controller, then specify that raw controller to the database manager using the CREATE TABLESPACE command:

```
CREATE TABLESPACE dms1
MANAGED BY DATABASE
USING (DEVICE '/dev/raw/raw1' 1170736)
```

Use the following information when working in a Linux environment. On Linux/390, the database manager does not support direct disk access devices.

Procedure

To configure raw I/O on Linux:

1. Calculate the number of 4 096-byte pages in this database partition, rounding down if necessary. For example:

```
# fdisk /dev/sda
Command (m for help): p

Disk /dev/sda: 255 heads, 63 sectors, 1106 cylinders
Units = cylinders of 16065 * 512 bytes
```

   In this example, the raw database partition to be used is `/dev/sda5`. It should not contain any valuable data.

Table 16. Linux raw I/O calculations.

Device boot	Start	End	Blocks	Id	System
/dev/sda1	1	523	4200997	83	Linux
/dev/sda2	524	1106	4682947+	5	Extended
/dev/sda5	524	1106	4682947	83	Linux

```
Command (m for help): q
#
```

   The number of pages in `/dev/sda5` is:

```
num_pages = floor( (4682947 * 1024)/4096 )
num_pages = 1170736
```

2. Create the table space by specifying the disk partition name. For example:

```
CREATE TABLESPACE dms1
MANAGED BY DATABASE
USING (DEVICE '/dev/sda5' 1170736)
```

3. To specify logical partitions by using junction points (or volume mount points), mount the RAW partition to another NTFS-formatted volume as a junction point, then specify the path to the junction point on the NTFS volume as the container path. For example:

```
CREATE TABLESPACE TS4
   MANAGED BY DATABASE USING (DEVICE 'C:\JUNCTION\DISK_1' 10000,
      DEVICE 'C:\JUNCTION\DISK_2' 10000)
```

   The database manager first queries the partition to see whether there is a file system R on it; if yes, the partition is not treated as a RAW device, and performs normal file system I/O operations on the partition.

   Table spaces on raw devices are also supported for all other page sizes supported by the database manager.

Altering table spaces

To alter a table space using the command line, use the ALTER TABLESPACE statement.

About this task

Depending on the type of table space, you can do things such as:

- Increasing the size of the table space by adding additional containers
- Resizing existing containers

- Dropping containers
- Rebalance the table space to start making use of new containers, or to move data out of dropped containers
- Lower the high water mark for the table space
- Reduce the overall size of the table space.

You can also rename a table space, and switch it from offline to online mode.

Calculating table space usage

You can determine how much of your table space is currently in use with the MON_GET_TABLESPACE table function. The information this function returns can help you determine whether you should attempt to reclaim free storage.

About this task

This task will provide you with information that you can use to determine the extent to which you have unused space below the high water mark for your table space. Based on this, you can make a determination as to whether reclaiming free storage would be beneficial.

Restrictions

Although you can determine various usage attributes about all your table spaces, only table spaces created with DB2 Version 9.7 or later have the reclaimable storage capability. If you want to be able to reclaim storage in table spaces created with earlier versions of the DB2 database product, you either must unload then reload the data into a table space created with DB2 Version 9.7, or move the data with an online move.

Procedure

To determine how much free space exists below the high water mark:

1. Formulate a SELECT statement that incorporates the MON_GET_TABLESPACE table function to report on the state of your table spaces. For example, the following statement will display the total pages, free pages, used pages, for all table spaces, across all database partitions:

```
SELECT varchar(tbsp_name, 30) as tbsp_name,
  reclaimable_space_enabled,
  tbsp_free_pages,
  tbsp_page_top,
  tbsp_usable_pages
FROM TABLE(MON_GET_TABLESPACE('',-2)) AS t
ORDER BY tbsp_free_pages ASC
```

2. Run the statement. You will see output that resembles this:

```
TBSP_NAME                      RECLAIMABLE_SPACE_ENABLED TBSP_FREE_PAGES      TBSP_PAGE_TOP        TBSP_USABLE_PAGES
------------------------------ ------------------------- -------------------- -------------------- --------------------
TEMPSPACE1                                             0                    0                    0                    1
SYSTOOLSTMPSPACE                                       0                    0                    0                    1
TBSP1                                                  1                    0                 1632                 1632
SMSDEMO                                                0                    0                    0                    1
SYSCATSPACE                                            1                 2012                10272                12284
USERSPACE1                                             1                 2496                 1696                 4064
IBMDB2SAMPLEREL                                        1                 3328                  736                 4064
TS1                                                    1                 3584                  480                 4064
TS2                                                    1                 3968                   96                 4064
TBSP2                                                  1                 3968                   96                 4064
```

```
TBSAUTO                                  1                3968                96                4064
SYSTOOLSPACE                             1                3976               116                4092

  12 record(s) selected.
```

3. Use the following formula to determine the number of free pages below the high water mark:

   freeSpaceBelowHWM = tbsp_free_pages - (tbsp_usable_pages - tbsp_page_top)

Results

Using the information from the report in step 2 on page 198, the free space below the high water mark for USERSPACE1 would be 2496 - (4064 - 1696) = 128 pages. This represents just slightly over 5% of the total free pages available in the table space.

What to do next

In this case, it might not be worth trying to reclaim this space. However, if you did want to reclaim those 128 pages, you could run an ALTER TABLESPACE USERSPACE1 REDUCE MAX statement. If you were to do so, and then run the MON_GET_TABLESPACE table function again, you would see the following:

```
TBSP_NAME                      RECLAIMABLE_SPACE_ENABLED TBSP_FREE_PAGES      TBSP_PAGE_TOP        TBSP_USABLE_PAGES
------------------------------ ------------------------- ------------------- -------------------- --------------------
TEMPSPACE1                                             0                   0                    0                    1
USERSPACE1                                             1                   0                 1568                 1568
SYSTOOLSTMPSPACE                                       0                   0                    0                    1
TBSP1                                                  1                   0                 1632                 1632
SMSDEMO                                                0                   0                    0                    1
SYSCATSPACE                                            1                2012                10272                12284
IBMDB2SAMPLEREL                                        1                3328                  736                 4064
TS1                                                    1                3584                  480                 4064
TS2                                                    1                3968                   96                 4064
TBSP2                                                  1                3968                   96                 4064
TBSAUTO                                                1                3968                   96                 4064
SYSTOOLSPACE                                           1                3976                  116                 4092

  12 record(s) selected.
```

Altering SMS table spaces

You cannot add containers to or change the size of containers for SMS table spaces once they have been created, with one exception; when you add new data partitions, you can add new containers to an SMS table space for those partitions.

Altering DMS table spaces

For DMS table spaces, you can add, extend, rebalance, resize, drop, or reduce containers.

Adding DMS containers

You can increase the size of a DMS table space (that is, one created with the MANAGED BY DATABASE clause) by adding one or more containers to the table space.

About this task

No rebalancing occurs if you are adding new containers and creating a new stripe set. A new stripe set is created using the BEGIN NEW STRIPE SET clause on the ALTER TABLESPACE statement. You can also add containers to existing stripe sets using the ADD TO STRIPE SET clause on the ALTER TABLESPACE statement.

The addition or modification of DMS containers (both file and raw device containers) is performed in parallel through prefetchers. To achieve an increase in parallelism of these create or resize container operations, you can increase the number of prefetchers running in the system. The only process which is not done in parallel is the logging of these actions and, in the case of creating containers, the tagging of the containers.

Note: To maximize the parallelism of the CREATE TABLESPACE or ALTER TABLESPACE statements (with respect to adding new containers to an existing table space) ensure that the number of prefetchers is greater than or equal to the number of containers being added. The number of prefetchers is controlled by the **num_ioservers** database configuration parameter. The database must be stopped for the new parameter value to take effect. In other words, all applications and users must disconnect from the database for the change to take effect.

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

The following example illustrates how to add two new device containers (each with 10 000 pages) to a table space on a Linux or UNIX operating system:

```
ALTER TABLESPACE RESOURCE
  ADD (DEVICE '/dev/rhd9'  10000,
       DEVICE  '/dev/rhd10' 10000)
```

Note that the ALTER TABLESPACE statement allows you to change other properties of the table space that can affect performance.

Dropping DMS containers

With a DMS table space, you can drop a container from the table space using the ALTER TABLESPACE statement.

About this task

Dropping a container is allowed only if the number of extents being dropped by the operation is less than or equal to the number of free extents above the high-water mark in the table space. This restriction is necessary because page numbers cannot be changed by the operation and therefore all extents up to and including the high-water mark must sit in the same logical position within the table space. Therefore, the resulting table space must have enough space to hold all of the data up to and including the high-water mark. In the situation where there is not enough free space, you receive an error immediately upon execution of the statement.

When containers are dropped, the remaining containers are renumbered such that their container IDs start at 0 and increase by 1. If all of the containers in a stripe set are dropped, the stripe set is removed from the map and all stripe sets following it in the map are shifted down and renumbered such that there are no gaps in the stripe set numbers.

Procedure

To drop a container, use the DROP option on the ALTER TABLESPACE statement.

Resizing DMS containers

Containers in a database managed (DMS) table space can be resized as storage needs change. If you use the auto-resize capabilities for DMS containers, the database manager handles this for you. If you did not enable the auto-resize option, you can also make adjustments manually.

About this task

To increase the size of one or more containers in a DMS table space by a specified amount, use the EXTEND option of the ALTER TABLESPACE command; To reduce the size of existing containers, use the REDUCE option. When you use EXTEND or REDUCE, you specify the amount by which you want to the size to increase or decrease from whatever it is currently. In other words, the size is adjusted relative to the current size.

You can also use the RESIZE option on the ALTER TABLESPACE statement. When you use RESIZE, you specify a new size for the affected containers. In other words, the size is interpreted as an absolute size for the specified containers. When using the RESIZE option, all of the containers listed as part of the statement must either be increased in size, or decreased in size. You cannot increase some containers and decrease other containers in the same statement.

The addition or modification of DMS containers (both file and raw device containers) is performed in parallel through prefetchers. To achieve an increase in parallelism of these create or resize container operations, you can increase the number of prefetchers running in the system. The only process which is not done in parallel is the logging of these actions and, in the case of creating containers, the tagging of the containers.

Note: To maximize the parallelism of the CREATE TABLESPACE or ALTER TABLESPACE statements (with respect to adding new containers to an existing table space) ensure that the number of prefetchers is greater than or equal to the number of containers being added. The number of prefetchers is controlled by the **num_ioservers** database configuration parameter. The database must be stopped for the new parameter value to take effect. In other words, all applications and users must disconnect from the database for the change to take effect.

Restrictions

- Each raw device can be used as only one container.
- The raw device size is fixed after its creation.
- When you are considering to use the RESIZE or EXTEND options to increase a raw device container, check the raw device size first to ensure that you do not attempt to increase the device container size larger than the raw device size.
- In DMS table spaces, a container must be at least two times the extent size pages in length. The maximum size of a container is operating system dependent.

Example

Example 1: Increasing the size of file containers. The following example illustrates how to increase file containers (each already existing with 1 000 pages) in a table space on a Windows operating system:

```
ALTER TABLESPACE PERSNEL
   EXTEND (FILE 'e:\wrkhist1'  200
           FILE  'f:\wrkhist2'  200)
```

The two files increase from 1 000 pages in size to 1 200 pages. The contents of the table space might be rebalanced across the containers. Access to the table space is not restricted during the rebalancing.

Example 2: Increasing the size of device containers. The following example illustrates how to increase two device containers (each already existing with 1 000 pages) in a table space on a Linux or UNIX operating system:

```
ALTER TABLESPACE HISTORY
  RESIZE (DEVICE '/dev/rhd7'  2000,
          DEVICE  '/dev/rhd8' 2000)
```

The two devices increase from 1 000 pages in size to 2 000 pages. The contents of the table space might be rebalanced across the containers. Access to the table space is not restricted during the rebalancing.

Example 3: Reducing container size using the REDUCE option. The following example illustrates how to reduce a file container (which exists with 1 000 pages) in a table space on a Windows operating system:

```
ALTER TABLESPACE PAYROLL
  REDUCE (FILE 'd:\hldr\finance'  200)
```

Following this action, the file is decreased from 1 000 pages in size to 800 pages.

Rebalancing DMS containers

The process of rebalancing involves moving table space extents from one location to another, and it is done in an attempt to keep data striped within the table space. You typically rebalance a table space when adding storage paths to or dropping storage paths from a database.

Effect of adding or dropping containers on rebalancing

When a table space is created, its table space map is created and all of the initial containers are lined up such that they all start in stripe 0. This means that data is striped evenly across all of the table space containers until the individual containers fill up. (See Example 1 (“Before”).)

Adding a container that is smaller than existing containers results in a uneven distribution of data. This can cause parallel I/O operations, such as prefetching data, to be performed less efficiently than they could on containers of equal size.

When new containers are added to a table space or existing containers are extended, a rebalance of the table space data will occur if the new space is added below the *high water mark* for the table space. If new space as added above the high water mark or if you are creating a new stripe set, a rebalance does not automatically occur. Rebalancing that is done to take advantage of added storage is known as a *forward rebalance*; in this case, the extent movement begins at extent 0 (the first extent in the table space) and proceeds upwards to the extent immediately below the high water mark.

Adding a container will almost always add space below the high-water mark, which is why a rebalance is often necessary when you add a container. You can force new containers to be added above the high-water mark, which allows you to choose not to rebalance the contents of the table space. An advantage of this

method is that the new container will be available for immediate use. Adding containers to a table space without rebalancing is done by adding a new *stripe set*. A stripe set is a set of containers in a table space that has data striped across it separately from the other containers that belong to that table space. The existing containers in the existing stripe sets remain untouched, and the containers you add become part of a new stripe set. To add containers without rebalancing, use the BEGIN NEW STRIPE SET clause on the ALTER TABLESPACE statement.

When containers are dropped from a table space, a rebalance automatically occurs if data resides in the space being dropped. In this case, the rebalance is known as a *reverse rebalance*; the extent movement begins at the high water mark and proceeds downwards to the first extent in the table space.

Before the rebalance starts, a new table space map is built based on the container changes made. The rebalancer will move extents from their location determined by the current map into the location determined by the new map.

Forward rebalancing

The rebalancer starts at extent 0, moving one extent at a time until the extent holding the high-water mark has been moved. As each extent is moved, the current map is altered, one piece at a time, to look like the new map. When the rebalance is complete, the current map and new map should look identical up to the stripe holding the high-water mark. The current map is then made to look completely like the new map and the rebalancing process is complete. If the location of an extent in the current map is the same as its location in the new map, then the extent is not moved and no I/O takes place.

When adding a new container, the placement of that container within the new map depends on its size and the size of the other containers in its stripe set. If the container is large enough such that it can start at the first stripe in the stripe set and end at (or beyond) the last stripe in the stripe set, then it will be placed that way (see Example 1 (“After”)). If the container is not large enough to do this, it will be positioned in the map such that it ends in the last stripe of the stripe set (see Example 3.) This is done to minimize the amount of data that needs to be rebalanced.

Access to the table space is not restricted during rebalancing; objects can be dropped, created, populated, and queried as usual. However, the rebalancing operation can have a significant impact on performance. If you need to add more than one container, and you plan to rebalance the containers, you should add them at the same time within a single ALTER TABLESPACE statement to prevent the database manager from having to rebalance the data more than once.

Note: In the following examples, the container sizes do not take the size of the container tag into account. The container sizes are very small, and are just used for the purpose of illustration, they are not recommended container sizes. The examples show containers of different sizes within a table space, but you are advised to use containers of the same size.

Reverse rebalancing

The rebalancer starts with the extent that contains the high-water mark, moving one extent at a time until extent 0 has been moved. As each extent is moved, the current map is altered one piece at a time to look like the new map. If the location of an extent in the current map is the same as its location in the new map, then the

extent is not moved and no I/O takes place.

Examples

Example 1 (before): Table space layout before containers added

If you create a table space with three containers and an extent size of 10, and the containers are 60, 40, and 80 pages which equate to 6, 4, and 8 extents, the table space is created with a map that can be diagrammed as shown in Figure 19.

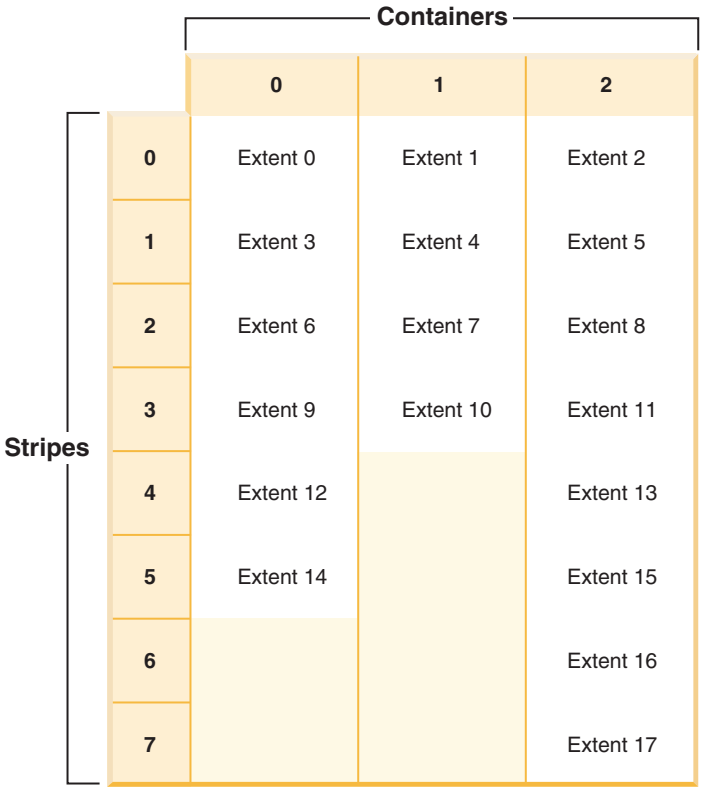


Figure 19. Table space with three containers and 18 extents

The corresponding table space map, as shown in a table space snapshot, looks like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0      11      119        0       3    0   3 (0, 1, 2)
   [1]       [0]       0      15      159        4       5    0   2 (0, 2)
   [2]       [0]       0      17      179        6       7    0   1 (2)
```

The headings in the table space map are Range Number, Stripe Set, Stripe Offset, Maximum extent number addressed by the range, Maximum page number addressed by the range, Start Stripe, End Stripe, Range adjustment, and Container list.

Example 1 (after): Adding a container that results in a forward rebalance being performed

If an 80-page container is added to the table space in Example 1, the container is large enough to start in the first stripe (stripe 0) and end in the last stripe (stripe 7). It is positioned such that it starts in the first stripe. The resulting table space can

be diagrammed as shown in Figure 20.

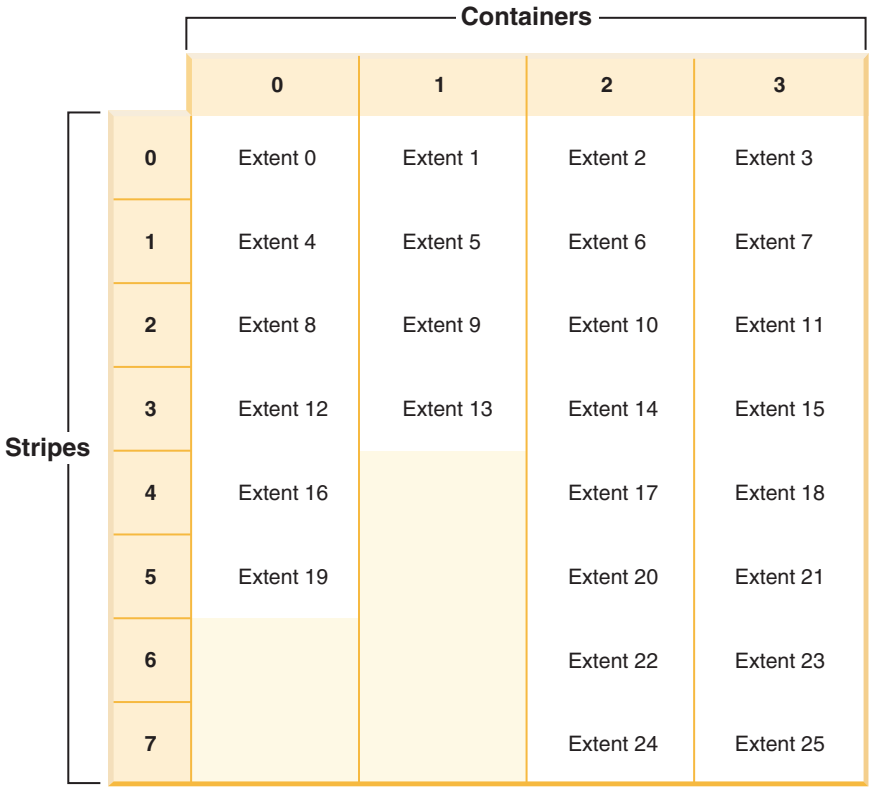


Figure 20. Table space with four containers and 26 extents

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]      0      15      159       0        3    0   4 (0, 1, 2, 3)
   [1]       [0]      0      21      219       4        5    0   3 (0, 2, 3)
   [2]       [0]      0      25      259       6        7    0   2 (2, 3)
```

If the high-water mark is within extent 14, the rebalancer starts at extent 0 and moves all of the extents up to and including 14. The location of extent 0 within both of the maps is the same so this extent is not required to move. The same is true for extents 1 and 2. Extent 3 does need to move so the extent is read from the old location (second extent within container 0) and is written to the new location (first extent within container 3). Every extent after this up to and including extent 14 is moved. Once extent 14 is moved, the current map looks like the new map and the rebalancer terminates.

If the map is altered such that all of the newly added space comes after the high-water mark, then a rebalance is not necessary and all of the space is available immediately for use. If the map is altered such that some of the space comes after the high-water mark, then the space in the stripes above the high-water mark is available for use. The rest is not available until the rebalance is complete.

If you decide to extend a container, the function of the rebalancer is similar. If a container is extended such that it extends beyond the last stripe in its stripe set,

the stripe set will expand to fit this and the following stripe sets will be shifted out accordingly. The result is that the container will not extend into any stripe sets following it.

Example 2: Extending a container

Consider the table space from Example 1. If you extend container 1 from 40 pages to 80 pages, the new table space looks like Figure 21.

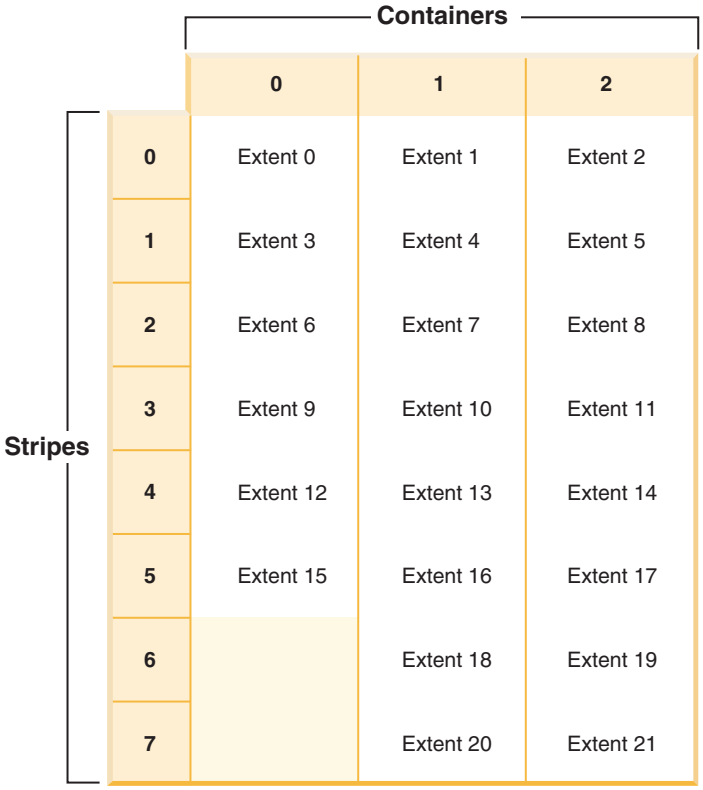


Figure 21. Table space with three containers and 22 extents

The corresponding table space map, as shown in a table space snapshot, looks like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0      17      179        0        5    0   3 (0, 1, 2)
   [1]       [0]       0      21      219        6        7    0   2 (1, 2)
```

Example 3: Adding a container not large enough to both start in the first stripe and end in the last

Consider the table space from Example 1. If a 50-page (5-extent) container is added to it, the container will be added to the new map in the following way. The container is not large enough to start in the first stripe (stripe 0) and end at or beyond the last stripe (stripe 7), so it is positioned such that it ends in the last stripe. (See Figure 22 on page 207.)

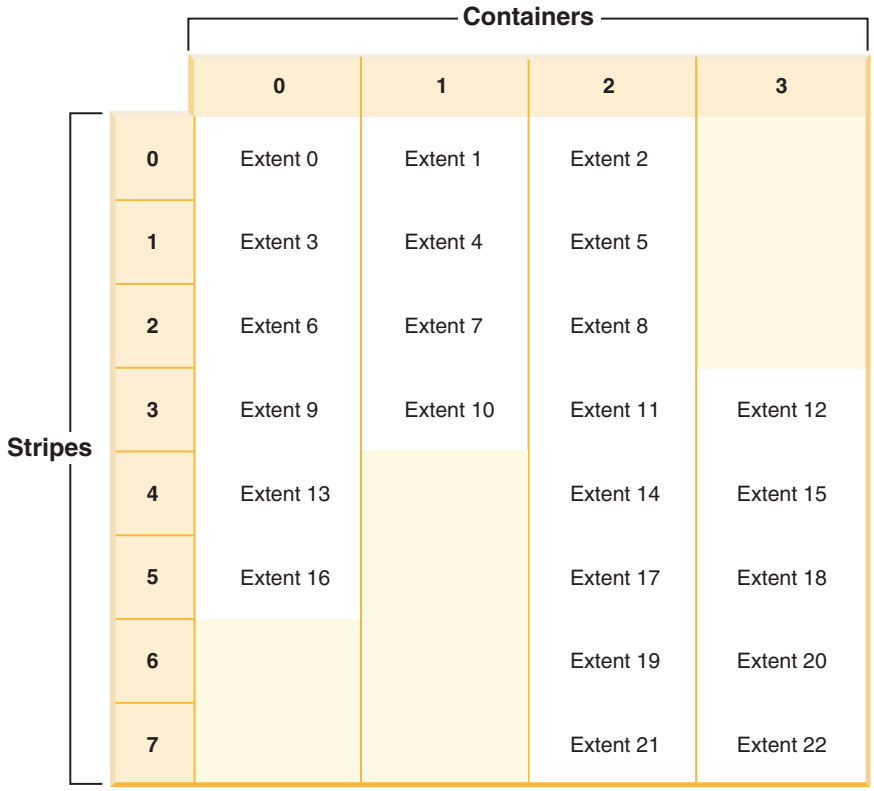


Figure 22. Table space with four containers and 23 extents

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]      [0]      0       8       89       0        2    0   3 (0, 1, 2)
   [1]      [0]      0      12      129       3        3    0   4 (0, 1, 2, 3)
   [2]      [0]      0      18      189       4        5    0   3 (0, 2, 3)
   [3]      [0]      0      22      229       6        7    0   2 (2, 3)
```

To extend a container, use the EXTEND or RESIZE clause on the ALTER TABLESPACE statement. To add containers and rebalance the data, use the ADD clause on the ALTER TABLESPACE statement. If you are adding a container to a table space that already has more than one stripe set, you can specify which stripe set you want to add to. To do this, you use the ADD TO STRIPE SET clause on the ALTER TABLESPACE statement. If you do not specify a stripe set, the default behavior will be to add the container to the current stripe set. The current stripe set is the most recently created stripe set, not the one that last had space added to it.

Any change to a stripe set might cause a rebalance to occur to that stripe set and any others following it.

You can monitor the progress of a rebalance by using table space snapshots. A table space snapshot can provide information about a rebalance such as the start time of the rebalance, how many extents have been moved, and how many extents must move.

Example 4: Dropping a container that results in a reverse rebalance being performed

Note: In the following examples, the container sizes do not take the size of the container tag into account. The container sizes are very small, and are just used for the purpose of illustration, they are not recommended container sizes. The examples show containers of different sizes within a table space, but this is just for the purpose of illustration; you are advised to use containers of the same size.

For example, consider a table space with three containers and an extent size of 10. The containers are 20, 50, and 50 pages respectively (2, 5, and 5 extents). The table space diagram is shown in Figure 23.

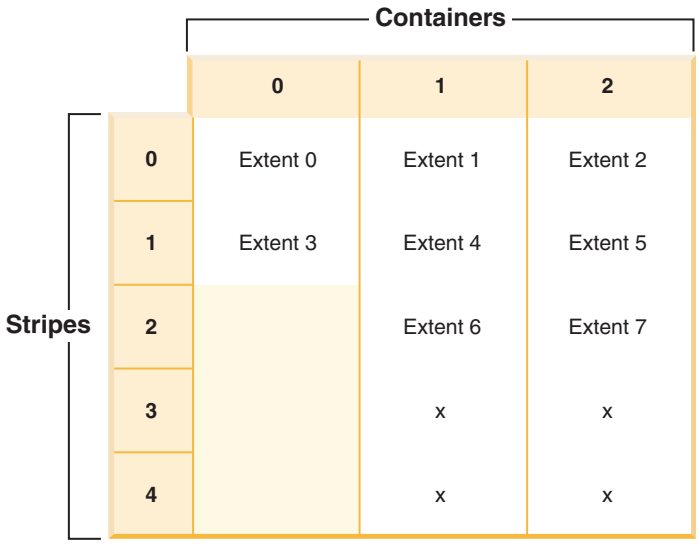


Figure 23. Table space with 12 extents, including four extents with no data

An X indicates that there is an extent but there is no data in it.

If you want to drop container 0, which has two extents, there must be at least two free extents above the high-water mark. The high-water mark is in extent 7, leaving four free extents, therefore you can drop container 0.

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0       5       59        0        1    0   3 (0, 1, 2)
   [1]       [0]       0      11      119        2        4    0   2 (1, 2)
```

After the drop, the table space will have just Container 0 and Container 1. The new table space diagram is shown in Figure 24 on page 209.

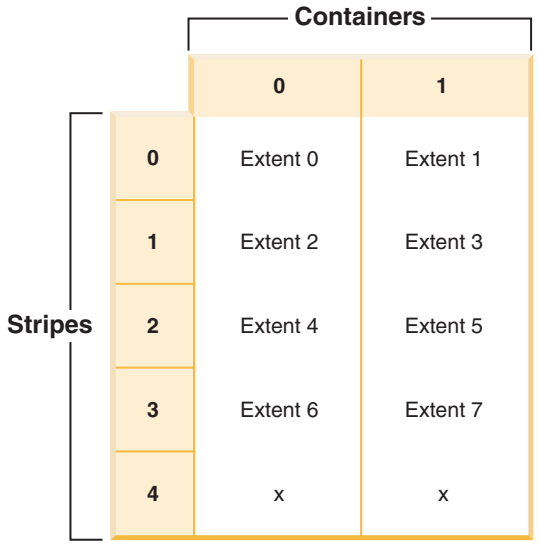


Figure 24. Table space after a container is dropped

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0       9       99        0        4    0   2 (0, 1)
```

Example 5: Adding a new stripe set

If you have a table space with three containers and an extent size of 10, and the containers are 30, 40, and 40 pages (3, 4, and 4 extents respectively), the table space can be diagrammed as shown in Figure 25.

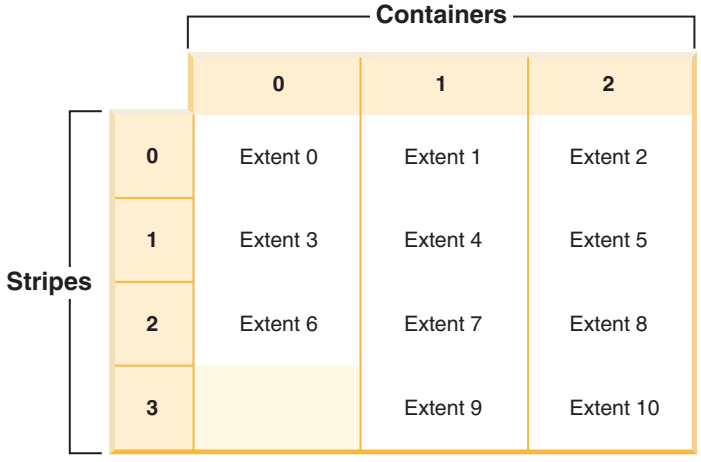


Figure 25. Table space with three containers and 11 extents

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0       8       89        0        2    0   3 (0, 1, 2)
   [1]       [0]       0      10      109        3        3    0   2 (1, 2)
```

When you add two new containers that are 30 pages and 40 pages (3 and 4 extents respectively) with the BEGIN NEW STRIPE SET clause, the existing ranges are not affected; instead, a new set of ranges is created. This new set of ranges is a stripe set and the most recently created one is called the current stripe set. After the two new containers is added, the table space looks like Figure 26.

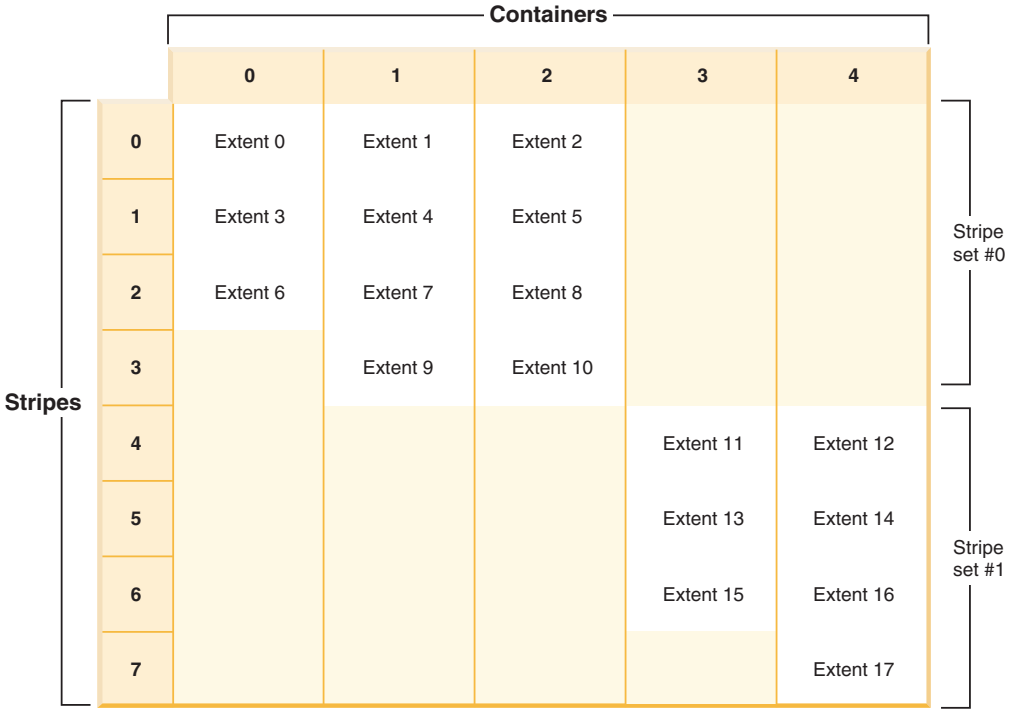


Figure 26. Table space with two stripe sets

The corresponding table space map, as shown in a table space snapshot, looks like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]      [0]      0       8       89       0        2    0   3 (0, 1, 2)
   [1]      [0]      0      10      109       3        3    0   2 (1, 2)
   [2]      [1]      4      16      169       4        6    0   2 (3, 4)
   [3]      [1]      4      17      179       7        7    0   1 (4)
```

If you add new containers to a table space, and you do *not* use the TO STRIPE SET clause with the ADD clause, the containers are added to the current stripe set (the highest stripe set). You can use the ADD TO STRIPE SET clause to add containers to any stripe set in the table space. You must specify a valid stripe set.

The database manager tracks the stripe sets using the table space map, and adding new containers without rebalancing generally causes the map to grow faster than when containers are rebalanced. When the table space map becomes too large, you will receive error SQL0259N when you try to add more containers.

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.
```

Reclaiming unused storage in DMS table spaces

You can reclaim unused storage in a DMS table space by telling the database manager to consolidate in-use extents lower in the table space. This also has the effect of lowering the high water mark. To reduce the container sizes in a DMS table space requires a separate REDUCE operation must also be performed.

Before you begin

You must have a DMS 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

To reclaim the unused storage in a DMS table space, you first must initiate an operation to cause extents in the table to be rearranged so as to make use of the free extents lower in the table space. This is done using the LOWER HIGH WATER MARK clause of the ALTER TABLESPACE statement. Next, you can reduce the size of the containers in the table space by a specified amount.

When you reduce the size of containers in a DMS table space, you must specify the names of the containers to reduce, or use the ALL CONTAINERS clause.

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

1. Use the ALTER TABLESPACE statement with the LOWER HIGH WATER MARK clause to reduce the high water mark as much as possible through the rearrangement of extents within the table space container.
2. Use the ALTER TABLESPACES statement with a REDUCE clause to reduce the size of some or all containers by a specified amount.

Example

Example 1: Lowering the high water mark, and reducing all containers by 5 megabytes. The following example lowers the high water mark for table space `ts`, and reduces the size of all containers in the table space by 5 megabytes.

```
ALTER TABLESPACE ts LOWER HIGH WATER MARK
ALTER TABLESPACE ts REDUCE (ALL CONTAINERS 5 M)
```

Example 2: Lowering the high water mark, and reducing container "Container1" by 2 000 pages. The following example lowers the high water mark for table space `ts`, and reduces the size of "Container1" by 2000 pages..

```
ALTER TABLESPACE ts LOWER HIGH WATER MARK
ALTER TABLESPACE ts REDUCE (FILE "Container1" 2000)
```

Prefetch size adjustment when adding or dropping containers

The default size for all prefetches from disk is set automatically for any table spaces created using DB2 versions 8.2 and later. This means that the database manager calculates a suitable prefetch size based on several factors, including the extent size, the number of containers in your table space, and the properties of your storage devices.

The degree to which prefetches of data can take place in parallel is a function of, among other things, the number of containers in a table space. For example, if you have two or more containers, then prefetches from each container can happen in parallel, which can improve overall database performance. If you change the number of containers in a table space by adding or dropping containers, the amount of data that you can efficiently prefetch might change. For example if you add a container, but the number of extents prefetched remains unchanged, then you might not be taking advantage of the opportunity to fetch additional data from the new container in parallel with that from the other containers. As containers are added or dropped, adjusting the prefetch size accordingly can maintain or improve performance by making I/O happen more efficiently.

You can set the prefetch size for table spaces manually, but once you do so, you must ensure that you update it as you change the containers in your table space if you want to maintain optimal prefetch performance. You can eliminate the need to update the prefetch size manually by setting PREFETCHSIZE for the table space to `AUTOMATIC` when using the CREATE TABLESPACE or ALTER TABLESPACE statements. AUTOMATIC is the default value for PREFETCHSIZE, unless you have modified the default value for the **dft_prefetch_sz** configuration parameter.

If you want to manually specify the prefetch size, you can do so in three ways:

- Create the table space with a specific prefetch size. If you manually choose a value for the prefetch size, you need to remember to adjust the prefetch size whenever there is an adjustment in the number of containers associated with the table space.
- When the **dft_prefetch_sz** database configuration parameter set to a value other than the default value of `AUTOMATIC`, omit the prefetch size when creating the table space. The database manager checks this parameter when there is no explicit mention of the prefetch size when creating the table space. If a value other than `AUTOMATIC` is found, then that value is what is used as the default prefetch size. You need to remember to adjust, if necessary, the prefetch size whenever there is an adjustment in the number of containers associated with the table space.
- Alter the prefetch size manually by using the ALTER TABLESPACE statement.

When manually adjusting the prefetch size, specify a size that corresponds to a disk stripe for optimal I/O parallelism. To calculate the prefetch size manually, use the formula:

number_of_containers × *number_of_disks_per_container* × *extent_size*

For example, assume the extent size for a database is 8 pages, and that there are 4 containers, each of which exists on a single physical disk. Setting the prefetch size to: 4 × 1 × 8 = 32 results in a prefetch size of 32 pages in total. These 32 pages will be read from each of the 4 containers in parallel.

If you have more than one physical disk per container, as you might if each container is made up of a RAID array, then to optimize I/O parallelism, ensure that the **DB2_PARALLEL_IO** registry variable is set correctly. (See "Parallel I/O for table space containers that use multiple physical disks" on page 214.) As you add or drop containers, if the prefetch size has been set manually, remember to update it to reflect an appropriate prefetch size. For example, assume each of 4 containers resides on a RAID 4+1 array, and the **DB2_PARALLEL_IO** registry variable has been

set to allow for parallel prefetches from each physical disk. Assume also an extent size of 8 pages. To read in one extent per container, you would set the prefetch size to $4 \times 4 \times 8 = 128$ pages.

Parallel I/O for table space containers that use multiple physical disks

Before the prefetch requests are submitted to the prefetch queues, they are broken down into a number of smaller, parallel prefetch requests, based on the number of containers in a table space. The **DB2_PARALLEL_IO** registry variable is used to manually override the parallelism of prefetch requests. (This is sometimes referred to as the *parallelism of the table space*). When **DB2_PARALLEL_IO** is set to NULL, which is the default, the parallelism of a table space is equal to the number of containers in the table space. If this registry variable is turned on, it defines the number of physical disks per container; the parallelism of a table space is equal to the number of containers multiplied by the value given in the **DB2_PARALLEL_IO** registry variable.

What follows are several other examples of how the **DB2_PARALLEL_IO** registry variable influences the parallelism of prefetches. Assume that table spaces have been defined with an AUTOMATIC prefetch size.

- **DB2_PARALLEL_IO**=NULL
  - Prefetching from table space containers is done in parallel, based on a combination of the following:
    - The number of containers in each table space
    - The size that was specified for prefetches on the CREATE or ALTER TABLESPACE statements, and in the **dft_prefetch_sz** configuration parameter.
  - Prefetches are not broken down into smaller, per-disk requests. If there are multiple physical disks associated with a container, prefetches from the disks for a single container will not take place in parallel.
  -
- **DB2_PARALLEL_IO**=*
  - All table spaces use the default number of spindles (6) for each container. The prefetch size is 6 times larger with parallel I/O on.
  - All table spaces have parallel I/O on. The prefetch request is broken down to several smaller requests, each equal to the prefetch size divided by the extent size (or equal to the number of containers times the number of spindles).
- **DB2_PARALLEL_IO**=*:3
  - All table spaces use 3 as the number of spindles per container.
  - All table spaces have parallel I/O on.
- **DB2_PARALLEL_IO**=*:3,1:1
  - All table spaces use 3 as the number of spindles per container except for table space 1, which uses 1.
  - All table spaces have parallel I/O on.

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.

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
   ```

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 27 illustrates a storage path being added, with the "before" and "after" layout of a rebalanced table space:

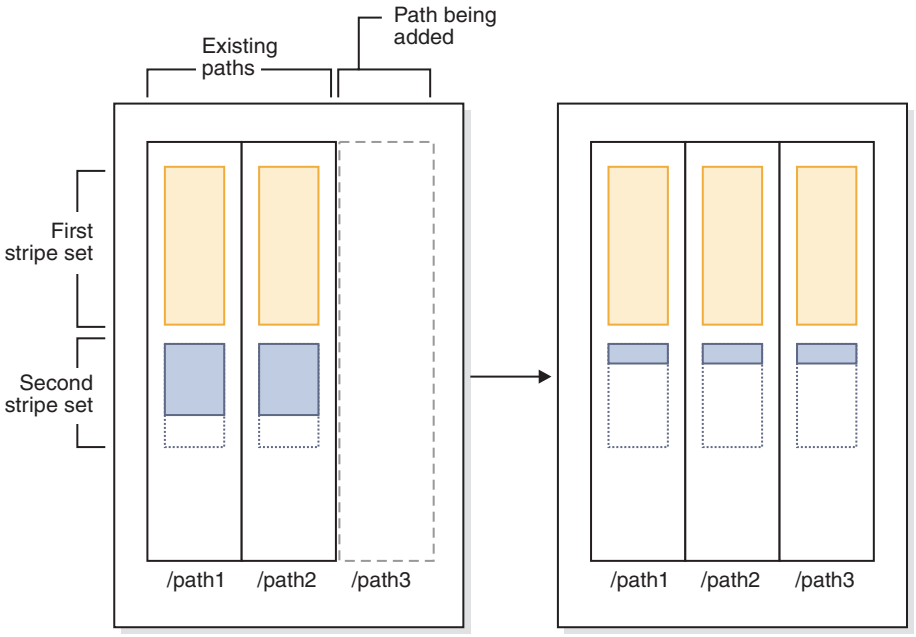


Figure 27. Adding a storage path and rebalancing an automatic storage table space

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 28 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.

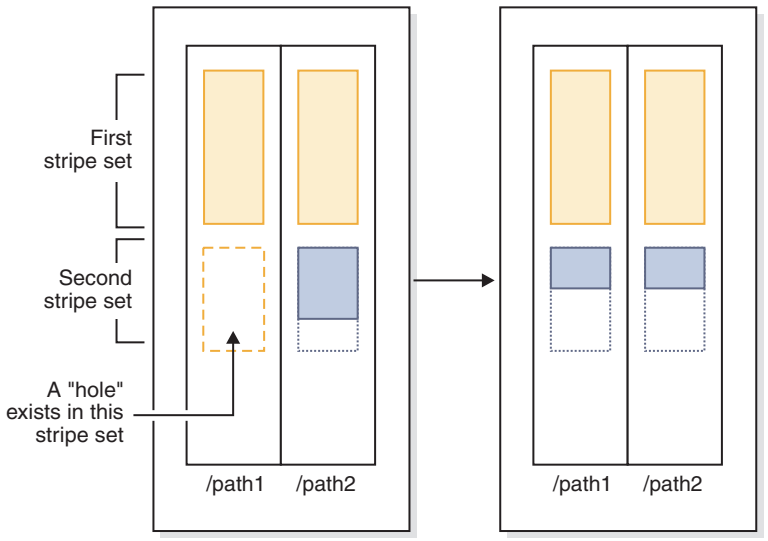


Figure 28. Rebalancing an automatic storage table space to fill gaps

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 29 on page 223 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:

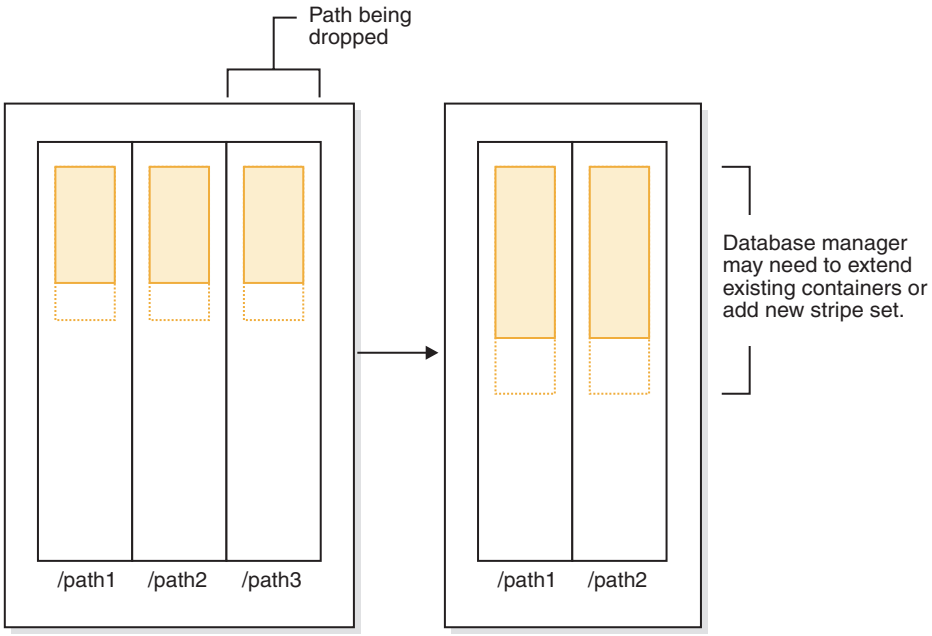


Figure 29. Dropping a storage path and rebalancing an automatic storage table space

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 220, 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 30 on page 225 is an example of storage paths being added and dropped, with the "before" and "after" layout of a rebalanced table space:

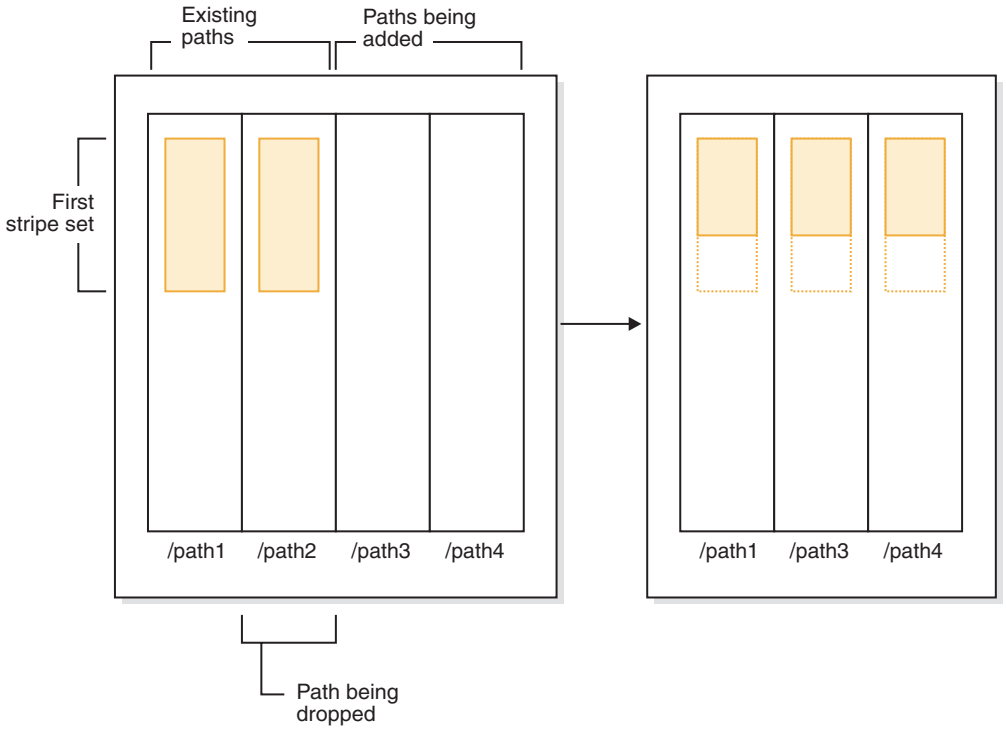


Figure 30. Adding and dropping storage paths, and then rebalancing an automatic storage table space

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.

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.

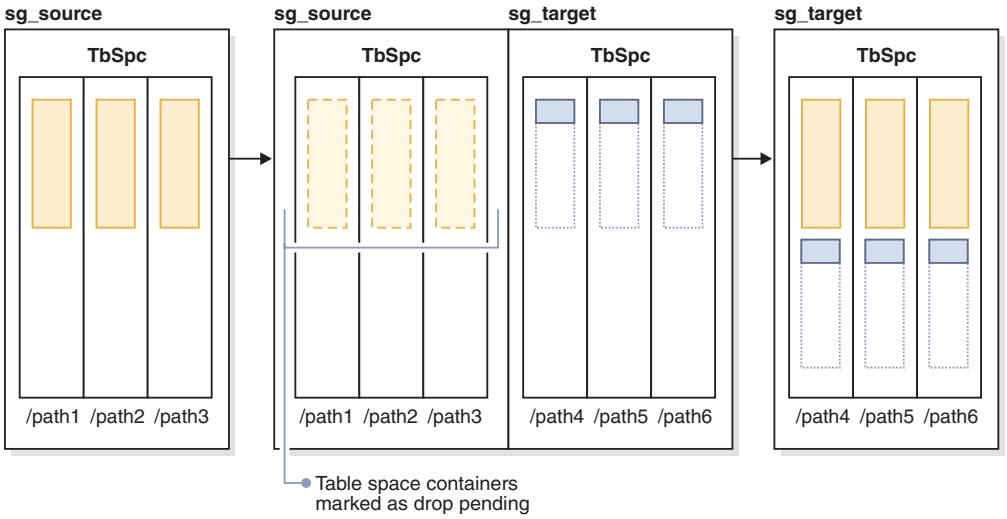


Figure 31. Moving a table space to a new storage group

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
```

Renaming a table space

Use the RENAME TABLESPACE statement to rename a table space.

About this task

You cannot rename the SYSCATSPACE table space. You cannot rename a table space that is in a rollforward pending or rollforward-in-progress state.

When restoring a table space that has been renamed since it was backed up, you must use the new table space name in the **RESTORE DATABASE** command. If you use the previous table space name, it will not be found. Similarly, if you are rolling

forward the table space with the **ROLLFORWARD DATABASE** command, ensure that you use the new name. If the previous table space name is used, it will not be found.

You can give an existing table space a new name without being concerned with the individual objects within the table space. When renaming a table space, all the catalog records referencing that table space are changed.

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 17. 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.

Table 17. Supported table space states (continued)

State	Hexadecimal state value	Description

Backup in Progress	0x800	This is a transient state that is only in effect during a backup operation.
		
		Example
		
		Perform and 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.

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 17. 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 17. 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 17. 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 17. 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 17. 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 17. 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.

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 18. 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 19. 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 Version 10.1, 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.

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.

Optimizing table space performance when data is on RAID devices

Follow these guidelines to optimize performance when data is stored on Redundant Array of Independent Disks (RAID) devices.

About this task

1. When creating a table space on a set of RAID devices, create the containers for a given table space (SMS or DMS) on separate devices.

   Consider an example where you have fifteen 146 GB disks configured as three RAID-5 arrays with five disks in each array. For RAID-5, the space of one disk is used for parity information, which is spread over all disks in the array. Hard drive manufacturers define a gigabyte as 1,000,000,000 bytes, but most operating systems use the 2-based notation, which explains why a 146 GB disk shows up as a 136 GB disk. After initializing the RAID-5 arrays, each array can store approximately 544 GB [1]. If you have a table space that requires 300 GB of storage, create three containers, and put each container on a separate array. Each container uses 100 GB (`300 GB/3`) on a device, and there are 444 GB (`544 GB - 100 GB`) left on each device for additional table spaces.

   Note: [1] `(146,000,000,000 bytes/1024/1024/1024) = 135.97 GB, (5*136 GB - 1*136 GB) = 544 GB`. Depending on the file system, which is placed on the array, additional space might be used for meta data and for the internal file system structure.
2. Select an appropriate extent size for the table spaces. The extent size for a table space is the amount of data that the database manager writes to a container before writing to the next container. Ideally, the extent size should be a multiple of the underlying segment size of the disks, where the segment size is the amount of data that the disk controller writes to one physical disk before writing to the next physical disk. Choosing an extent size that is a multiple of the segment size ensures that extent-based operations, such as parallel sequential read in prefetching, do not compete for the same physical disks. Also, choose an extent size that is a multiple of the page size.

   In the example, if the segment size is 64 KB and the page size is 16 KB, an appropriate extent size might be 256 KB.
3. Use the DB2_PARALLEL_IO registry variable to enable parallel I/O for all table spaces and to specify the number of physical disks per container.

   For the situation in the example, set `DB2_PARALLEL_IO = *:5`.

   If you set the prefetch size of a table space to AUTOMATIC, the database manager uses the number of physical disks value that you specified for DB2_PARALLEL_IO to determine the prefetch size value. If the prefetch size is not set to AUTOMATIC, you can set it manually, taking into account the RAID stripe size, which is the value of the segment size multiplied by the number of active disks. Choose a prefetch size value that meets the following conditions:

   - It is equal to the RAID stripe size multiplied by the number of RAID parallel devices (or a whole number representation of this product).
   - It is a whole number representation of the extent size.

   In the example, you might set the prefetch size to 768 KB. This value is equal to the RAID stripe size (256 KB) multiplied by the number of RAID parallel

devices (3). It is also a multiple of the extent size (256 KB). Choosing this prefetch size means that a single prefetch will engage all the disks in all the arrays. If you want the prefetchers to work more aggressively because your workload involves mainly sequential scans, you can instead use a multiple of this value, such as 1536 KB (768 KB x 2).

4. Do not set the DB2_USE_PAGE_CONTAINER_TAG registry variable. As described earlier, you should create a table space with an extent size that is equal to, or a multiple of, the RAID stripe size. However, when you set DB2_USE_PAGE_CONTAINER_TAG to ON, a one-page container tag is used, and the extents do not line up with the RAID stripes. As a result, it might be necessary during an I/O request to access more physical disks than would be optimal.

Procedure

- When creating a table space on a set of RAID devices, create the containers for the table space (SMS or DMS) on separate devices.

  Consider an example where you have fifteen 146 GB disks configured as three RAID-5 arrays with five disks in each array. After formatting, each disk can hold approximately 136 GB of data. Each array can therefore store approximately 544 GB (4 active disks x 136 GB). If you have a table space that requires 300 GB of storage, create three containers, and put each container on a separate device. Each container uses 100 GB (300 GB/3) on a device, and there are 444 GB (544 GB - 100 GB) left on each device for additional table spaces.

- Select an appropriate extent size for the table spaces.

  The extent size for a table space is the amount of data that the database manager writes to a container before writing to the next container. Ideally, the extent size should be a multiple of the underlying segment size of the disks, where the segment size is the amount of data that the disk controller writes to one physical disk before writing to the next physical disk. Choosing an extent size that is a multiple of the segment size ensures that extent-based operations, such as parallel sequential read in prefetching, do not compete for the same physical disks. Also, choose an extent size that is a multiple of the page size.

  In the example, if the segment size is 64 KB and the page size is 16 KB, an appropriate extent size might be 256 KB.

- Use the **DB2_PARALLEL_IO** registry variable to enable parallel I/O for all table spaces and to specify the number of physical disks per container.

  For the situation in the example, set **DB2_PARALLEL_IO** = `*:4`.

  If you set the prefetch size of a table space to AUTOMATIC, the database manager uses the number of physical disks value that you specified for **DB2_PARALLEL_IO** to determine the prefetch size value. If the prefetch size is not set to AUTOMATIC, you can set it manually, taking into account the RAID stripe size, which is the value of the segment size multiplied by the number of active disks. Choose a prefetch size value that meets the following conditions:

  – It is equal to the RAID stripe size multiplied by the number of RAID parallel devices (or a whole number representation of this product).

  – It is a whole number representation of the extent size.

  In the example, you might set the prefetch size to 768 KB. This value is equal to the RAID stripe size (256 KB) multiplied by the number of RAID parallel devices (3). It is also a multiple of the extent size (256 KB). Choosing this prefetch size means that a single prefetch engages all the disks in all the arrays. If you want the prefetchers to work more aggressively because your workload involves mainly sequential scans, you can instead use a multiple of this value, such as 1536 KB (768 KB x 2).

- Do not set the **DB2_USE_PAGE_CONTAINER_TAG** registry variable. As described earlier, you should create a table space with an extent size that is equal to, or a multiple of, the RAID stripe size. However, when you set **DB2_USE_PAGE_CONTAINER_TAG** to ON, a one-page container tag is used, and the extents do not line up with the RAID stripes. As a result, it might be necessary during an I/O request to access more physical disks than would be optimal.

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 9. 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 32 on page 245, 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.

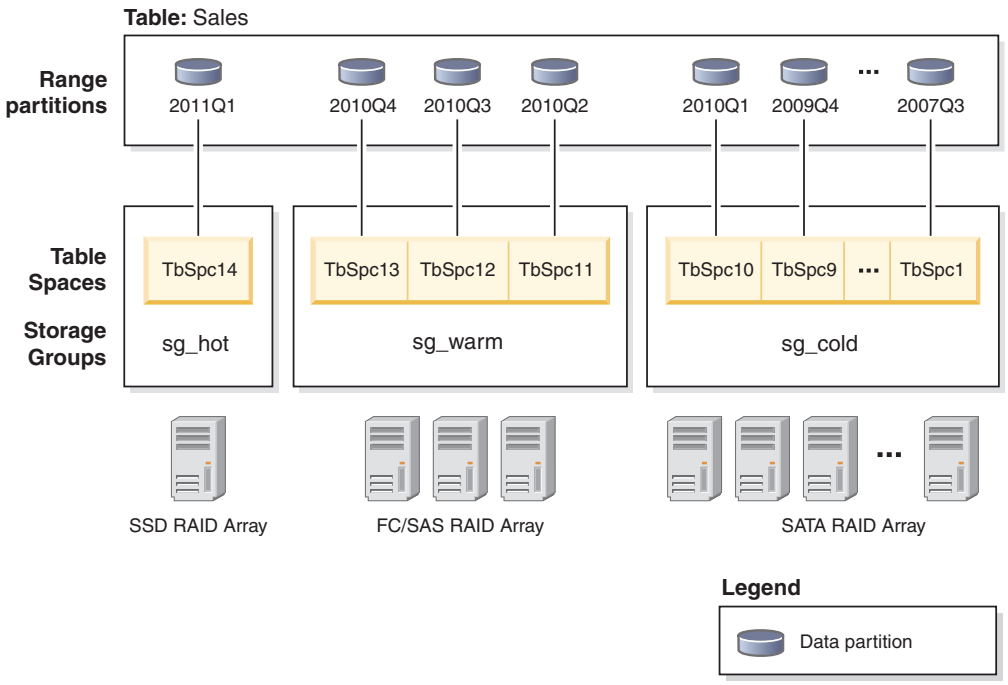


Figure 32. Managing Sales data using multi-temperature data storage

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 20. 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 21. 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 Version 10.1, 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.

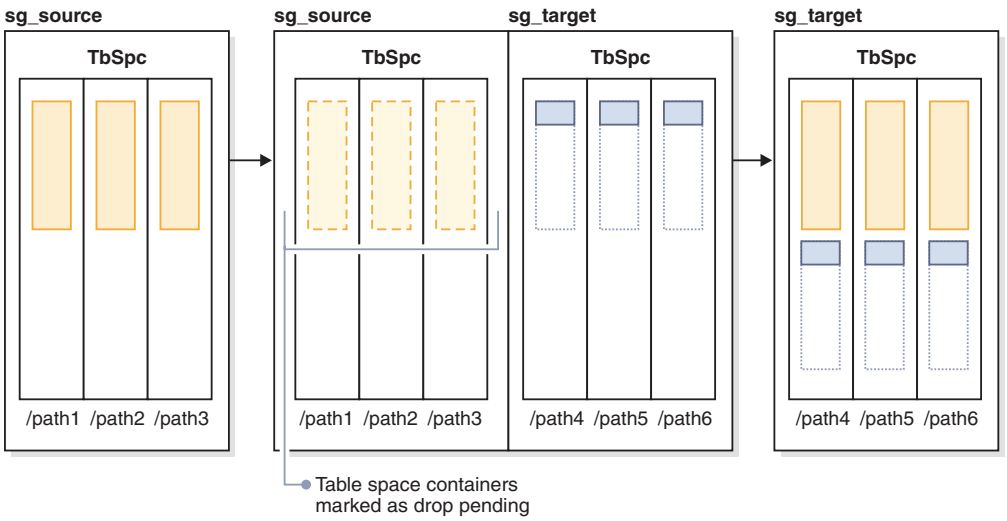


Figure 33. Moving a table space to a new storage group

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 10. 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.

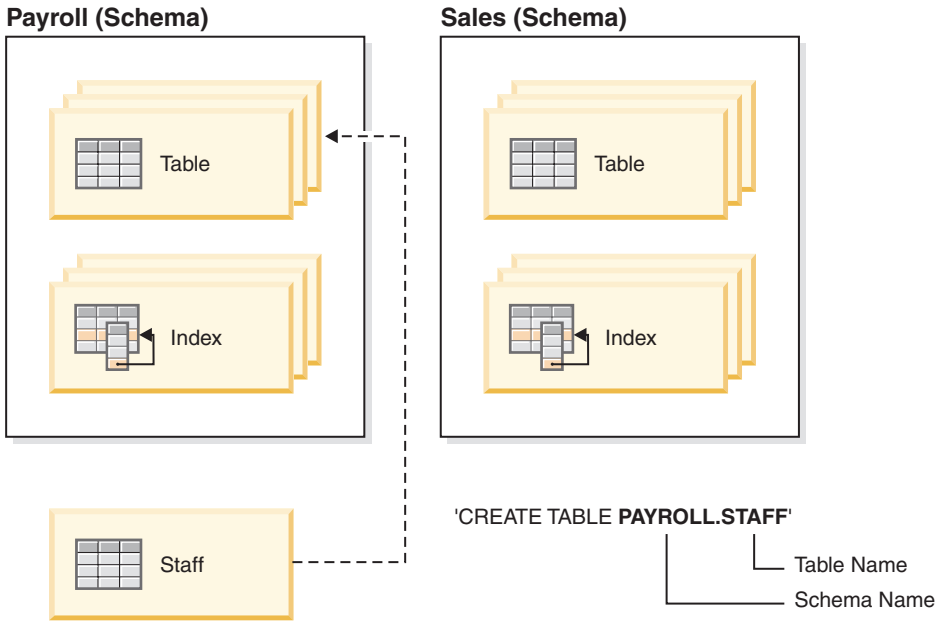
Schemas can be compared to directories, with the current schema being the current directory. Using this analogy, SET SCHEMA is equivalent to the **change directory** command.

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 261.

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 261.

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
```

Restarting a failed copy schema operation

Errors occurring during a **db2move** COPY operation can be handled in various ways depending on the type of object being copied, or the phase during which the COPY operation failed (that is, either the recreation of objects phase, or the loading of data phase).

About this task

The **db2move** utility reports errors and messages to the user using message and error files. Copy schema operations use the COPYSCHEMA_*timestamp*.MSG message file, and the COPYSCHEMA_*timestamp*.err error file. These files are created in the current working directory. The current time is appended to the file name to ensure uniqueness of the files. It is up to the user to delete these message and error files when they are no longer required.

Note: It is possible to have multiple **db2move** instances running simultaneously. The COPY option does not return any SQLCODES. This is consistent with **db2move** behavior.

The type of object being copied can be categorized as one of two types: physical objects and business objects.

A physical object refers to an object that physically resides in a container, such as tables, indexes and user-defined structured types. A business object refers to cataloged objects that do not reside in containers, such as views, user-defined structured types (UDTs), and aliases.

Errors occurring during the recreation of a physical object cause the utility to roll back, whereas, errors during the recreation of a logical object do not.

Procedure

To restart the copy schema operation:

After addressing the issues causing the load operations to fail (described in the error file), reissue the **db2move-COPY** command. Use the **-tf** parameter with the LOADTABLE.err file name to specify which tables to copy and populate with data. For example:

```
db2move sourcedb COPY -tf LOADTABLE.err -co TARGET_DB mytarget_db
  -mode load_only
```

You can also input the table names manually using the **-tn** parameter. For example:

```
db2move sourcedb COPY -tn "FOO"."TABLE1","FOO 1"."TAB 444",
  -co TARGET_DB mytarget_db -mode load_only
```

Note: The load_only mode requires inputting at least one table using the **-tn** or **-tf** parameter.

Examples

Example 1: Schema copy errors related to physical objects

Failures which occur during the recreation of physical objects on the target database, are logged in the error file COPYSCHEMA_timestamp.err. For each failing object, the error file contains information such as object name, object type, DDL text, time stamp, and a string formatted sqlca (sqlca field names, followed by their data values).

Sample output for the COPYSCHEMA_*timestamp*.err error file:

```
1. schema: FOO.T1
Type:      TABLE
Error Msg: SQL0104N An unexpected token 'FOO.T1'...
Timestamp: 2005-05-18-14.08.35.65
```

```
DDL:       create view FOO.v1

2. schema:  FOO.T3
Type:      TABLE
Error Msg: SQL0204N FOO.V1 is an undefined name.
Timestamp: 2005-05-18-14.08.35.68
DDL:       create table FOO.T3
```

If any errors creating physical objects are logged at the end of the recreation phase and before attempting the load phase, the **db2move** utility 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 source database. In order to gather all possible errors into the error file, the rollback occurs at the end of the recreation phase after attempting to re-create each object, rather than after the first failure. This allows you the opportunity to fix any problems before restarting the **db2move** operation. If there are no failures, the error file is deleted.

Example 2: Schema copy errors related to business objects

Failures that occur during the recreation of business objects on the target database, do not cause the **db2move** utility to fail. Instead, these failures are logged in the COPYSCHEMA_*timestamp*.err error file. Upon completion of the **db2move** utility, you can examine the failures, address any issues, and manually re-create each failed object (the DDL is provided in the error file for convenience).

If an error occurs when **db2move** is attempting to repopulate table data using the load utility, the **db2move** utility does not fail. Rather, generic failure information is logged to the COPYSCHEMA_*timestamp*.err file (for example, the object name, object type, DDL text, time stamp, and sqlca), and the fully qualified name of the table is logged into another file, LOADTABLE_*timestamp*.err. Each table is listed per line to satisfy the **db2move -tf** parameter format, similar to the following:

```
"FOO"."TABLE1"
"FOO 1"."TAB 444"
```

Example 3: Other types of db2move failures

Internal operations such as memory errors, or file system errors can cause the **db2move** utility to fail.

If the internal operation failure occurs during the DDL recreation phase, all successfully created objects are rolled back from the target schema. All internally created tables such as the DMT table and the **db2look** table, are cleaned up on the source database.

If the internal operation failure occurs during the load phase, all successfully created objects remain on the target schema. All tables that experience a failure during a load operation, and all tables which are not yet loaded, are logged in the LOADTABLE.err error file. You can then issue the **db2move COPY** command using the LOADTABLE.err as discussed in Example 2. If the **db2move** utility abends (for example a system crash occurs, the utility traps, or the utility is killed), then the information regarding which tables still must be loaded is lost. In this case, you can drop the target schema using the ADMIN_DROP_SCHEMA procedure and reissue the **db2move COPY** command.

Regardless of what error you might encounter during an attempted copy schema operation, you always have the option of dropping the target schema using the ADMIN_DROP_SCHEMA procedure. You can then reissue the **db2move COPY** command.

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
```

Part 3. Database objects

Physical database design consists of defining database objects and their relationships.

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 11. Concepts common to most database objects

Aliases

An *alias* is an alternative name for an object such as a module, table or another alias. It can be used to reference an object wherever that object can be referenced directly.

An alias cannot be used in all contexts; for example, it cannot be used in the check condition of a check constraint. An alias cannot reference a declared temporary table but it can reference a created temporary table.

Like other objects, an alias can be created, dropped, and have comments associated with it. Aliases can refer to other aliases in a process called *chaining* as long as there are no circular references. Aliases do not require any special authority or privilege to use them. Access to the object referred to by an alias, however, does require the authorization associated with that object.

If an alias is defined as a *public* alias, it can be referenced by its unqualified name without any impact from the current default schema name. It can also be referenced using the qualifier SYSPUBLIC.

Synonym is an alternative name for alias.

For more information, refer to "Aliases in identifiers" in the *SQL Reference Volume 1*.

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
```

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 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 12. 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 318.

Table design concepts

When designing tables, you must be familiar with some related concepts.

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 34 on page 280 shows the data types that are supported by DB2 databases.

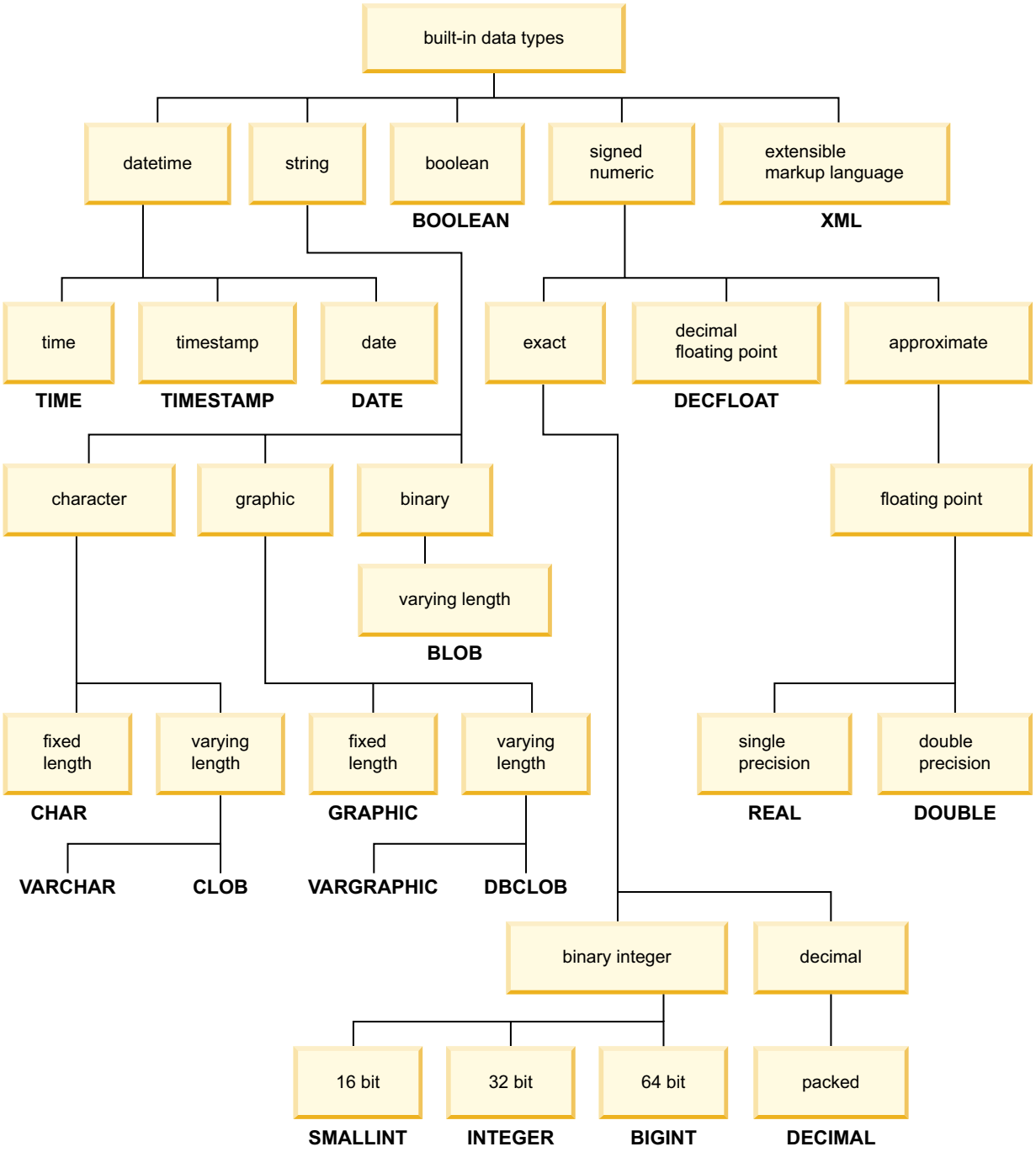


Figure 34. Built-in data types

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.

Column data 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.

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 13, "Constraints," on page 389 or "Types of constraints" on page 389.

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.

Primary key, referential integrity, check, and unique constraints

Constraints are rules that limit the values that can be inserted, deleted, or updated in a table.

Primary key constraints
: 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.

Referential integrity (or foreign key) constraints
: 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.

Check constraints
: A (table) check constraint sets restrictions on data added to a specific table.

Unique constraints
: 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.

Unicode table and data considerations

The Unicode character encoding standard is a fixed-length, character encoding scheme that includes characters from almost all of the living languages of the world.

For more information about Unicode table and data considerations, see:

- “Unicode character encoding” in *Globalization Guide*
- “Character comparisons based on collating sequences” in *Globalization Guide*
- “Date and time formats by territory code” in *Globalization Guide*
- “Conversion table files for euro-enabled code pages” in *Globalization Guide*

Additional information on Unicode can be found in the latest edition of *The Unicode Standard* , and from the Unicode Consortium website at www.unicode.org.

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 22. 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
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 called pages. Pages can be four sizes: 4, 8, 16, and 32 kilobytes. Table data pages do not contain the data for columns defined with LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DCLOB, or XML data types, unless the LOB or XML document is inlined through the use of 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 through the use of the INLINE LENGTH option of the CREATE and ALTER TABLE statements.

You can create buffer pools or table spaces that have page sizes of 4 KB, 8 KB, 16 KB, or 32 KB. 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 you are using an 8 KB, 16 KB, or 32 KB page size. You can have a maximum of 500 columns for a 4 KB page size. The maximum of rows you can have per page is 255, regardless of the page size.

Maximum row lengths vary, depending on page size used:

- When the page size is 4 KB, the row length can be up to 4 005 bytes.
- When the page size is 8 KB, the row length can be up to 8 101 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.

To determine the page size for a table space you must consider the following:

- For OLTP applications that perform random row read and write operations, a smaller page size is usually preferable, because it consumes 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 usually better, because it reduces the number of I/O requests that are required to read a specific number of rows. There is, however, an exception to this. If your row size is smaller than `pagesize / maximum rows`, there will be consumed space on each page. In this situation, a smaller page size might be more appropriate.

Larger page sizes might allow you to 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.

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 35 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.

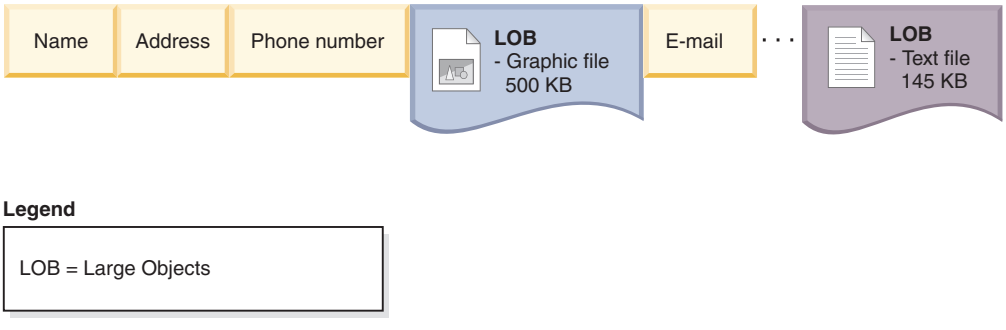


Figure 35. The problem of including LOB data within base table rows

To reduce the difficulties associated with working with LOBs, they are treated differently from other data types. Figure 36 on page 293, 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.

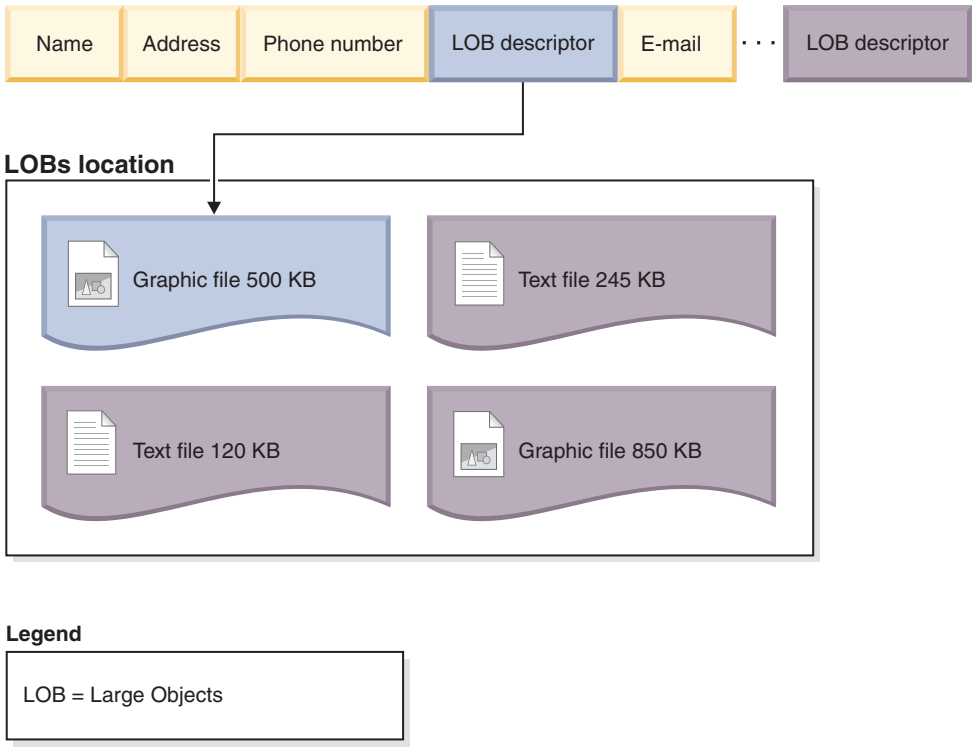


Figure 36. LOB descriptors within the base table row refer to the LOBs within the separate LOBs location

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 37 on page 294.

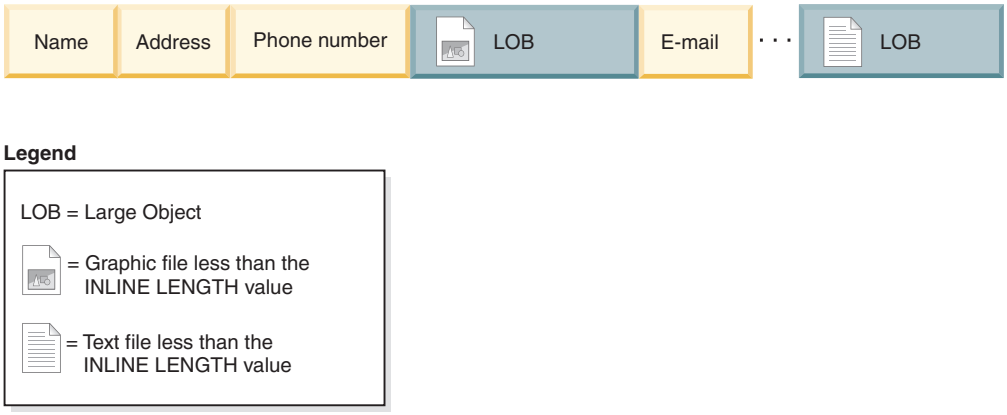


Figure 37. Small LOBs included within base table rows

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.

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 286".

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 168.

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 303.

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 286".

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 168.

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.

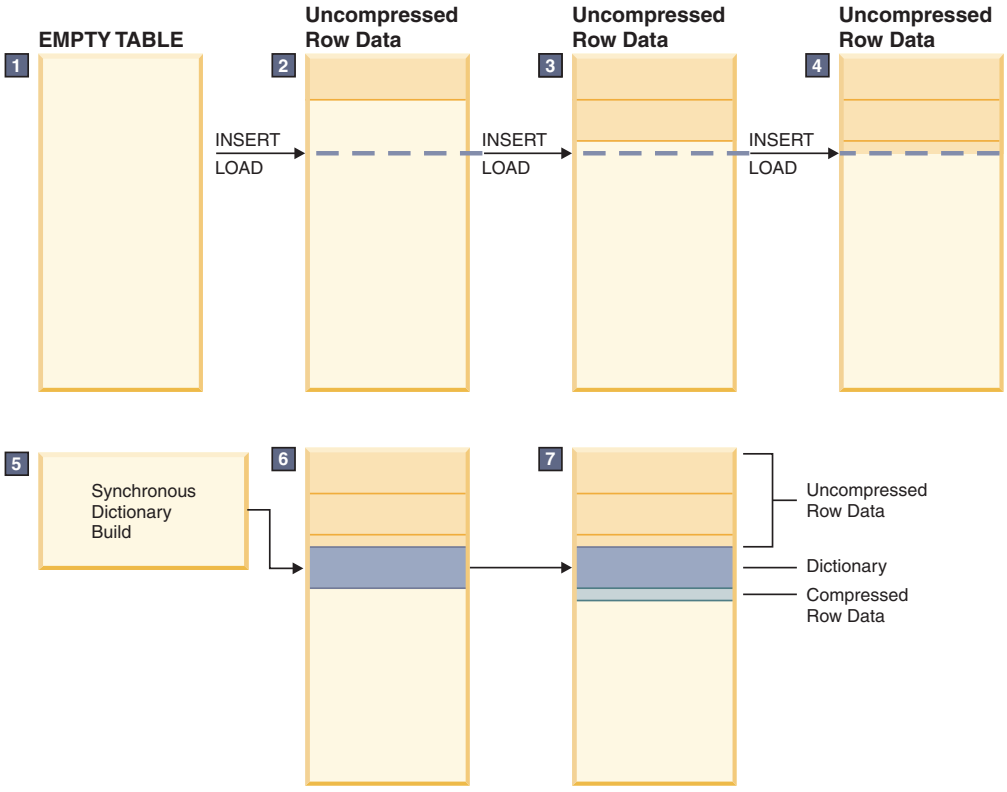
Automatic dictionary creation

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.

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.

Optimistic locking overview

Enhanced optimistic locking support provides a technique for SQL database applications that does not hold row locks between selecting, and updating or deleting rows.

Applications can be written to optimistically assume that unlocked rows are unlikely to change before the update or delete. If the rows do change, the updates or deletes will fail and the application's logic can handle such failures, for example, by retrying the select.

The advantage of this enhanced optimistic locking is improved concurrency, since other applications can read and write those same rows. In three-tier environments where business transactions have no correlation to database transactions, this optimistic locking technique is used, since locks cannot be maintained across business transactions.

Table 23 lists the relevant topics in each category.

Table 23. Overview to optimistic locking information

Category	Related topics
General information and restrictions	• "Optimistic locking" • "Granularity of row change tokens and false negatives" on page 312 • "Optimistic locking restrictions and considerations" on page 311
Time-based updates	• "Time-based update detection" on page 313 • "Time values generated for ROW CHANGE TIMESTAMPs" on page 314
Enabling	• "Planning the enablement of optimistic locking" on page 316 • "Enabling optimistic locking in applications" on page 317
Usage scenarios	• "Scenarios: Optimistic locking and time-based detection" on page 384 – "Scenario: Using optimistic locking in an application program" on page 384 – "Scenario: Time-based update detection" on page 387 – "Scenarios: Optimistic locking using implicitly hidden columns" on page 386
Reference information	• "RID_BIT() and RID() built-in function" on page 314 • ALTER TABLE statement in *SQL Reference Volume 1* • CREATE TABLE statement in *SQL Reference Volume 2* • DELETE statement in *SQL Reference Volume 2* • SELECT statement in *SQL Reference Volume 2* • UPDATE statement in *SQL Reference Volume 2*

Note: Throughout the optimistic locking topics, whenever a row is referred to as being inserted or updated, this refers to all forms of SQL statements that could cause a row to be inserted into a table or updated in any way. For instance, INSERT, UPDATE, MERGE, or even the DELETE statement (with referential constraints) can all cause the timestamp column to be either created or updated.

Optimistic locking

Optimistic locking is a technique for SQL database applications that does not hold row locks between selecting and updating or deleting a row.

The application is written to optimistically assume that unlocked rows are unlikely to change before the update or delete operation. If the row does change, the update or delete will fail and the application logic handles such failures by, for example, retrying the select. One advantage of optimistic locking is improved concurrency, because other applications can read and write that row. In a three tier environment where business transactions have no correlation to database transaction, the optimistic locking technique is used, because locks cannot be maintained across the business transaction.

However, optimistic locking by values has some disadvantages:

- Can result in *false positives* without additional data server support, a condition when using optimistic locking whereby a row that is *changed* since it was selected cannot be updated without first being selected again. (This can be

contrasted with *false negatives*, the condition whereby a row that is *unchanged* since it was selected cannot be updated without first being selected again.)

- Requires more re-try logic in applications
- It is complicated for applications to build the UPDATE search conditions
- It is inefficient for the DB2 server to search for the target row based on values
- Data type mismatches between some client types and database types, for example, timestamps, prevent all columns from being used in the searched update

The support for easier and faster optimistic locking with no *false positives* has the following new SQL functions, expressions, and features:

- Row Identifier (RID_BIT or RID) built-in function
- ROW CHANGE TOKEN expression
- Time-based update detection
- Implicitly hidden columns

DB2 applications can enable *optimistic locking by values* by building a searched UPDATE statement that finds the row with the exact same values that were selected. The searched UPDATE fails if the row's column values have changed.

Applications using this programming model will benefit from the enhanced optimistic locking feature. Note that applications that do not use this programming model are not considered optimistic locking applications, and they will continue to work as before.

Row Identifier (RID_BIT or RID) built-in function
This built-in function can be used in the SELECT list or predicates statement. In a predicate, for example, WHERE RID_BIT(tab)=?, the RID_BIT equals predicate is implemented as a new direct access method in order to to efficiently locate the row. Previously, so called values optimistic locking with values was done by adding all the selected column values to the predicates and relying on some unique column combinations to qualify only a single row, with a less efficient access method.

ROW CHANGE TOKEN expression
This new expression returns a token as BIGINT. The token represents a relative point in the modification sequence of a row. An application can compare the current ROW CHANGE TOKEN value of a row with the ROW CHANGE TOKEN value that was stored when the row was last fetched to determine whether the row has changed.

Time-based update detection:
This feature is added to SQL using the RID_BIT() and ROW CHANGE TOKEN. To support this feature, the table needs to have a new generated column defined to store the timestamp values. This can be added to existing tables using the ALTER TABLE statement, or the column can be defined when creating a new table. The column's existence, also affects the behavior of optimistic locking in that the column if it is used to improve the granularity of the ROW CHANGE TOKEN from page level to row level, which could greatly benefit optimistic locking applications. This feature has also been added to DB2 for z/OS.

Implicitly hidden columns:
For compatibility, this feature eases the adoption of the RID_BIT and ROW

CHANGE TOKEN columns to existing tables and applications. Implicitly hidden columns are not externalized when implicit column lists are used. For example:

- A SELECT * against the table does not return a implicitly hidden columns in the result table
- An INSERT statement without a column list does not expect a value for implicitly hidden columns, but the column should be defined to allow nulls or have another default value.

Note: Refer to the DB2 Glossary for the definition of optimistic locking terms, such as *optimistic concurrency control*, *pessimistic locking*, *ROWID*, and *update detection*.

Optimistic locking restrictions and considerations

This topic lists optimistic locking restrictions that you must be aware of.

- ROW CHANGE TIMESTAMP columns are not supported in the following keys, columns, and names (sqlstate 429BV is returned if used):
  - Primary keys
  - Foreign keys
  - Multidimensional clustering (MDC) columns
  - Table partitioning columns
  - Database hashed partitioning keys
  - DETERMINED BY constraint columns
  - Nicknames
- The RID() function is not supported in partitioned database configurations.
- Online or offline table reorg performed between the fetch and update operations in an optimistic locking scenario can cause the update to fail, but this should be handled by normal application re-try logic.
- The IMPLICITLY HIDDEN attribute is restricted to only ROW CHANGE TIMESTAMP columns for optimistic locking.
- Inplace reorg is restricted for tables where a ROW CHANGE TIMESTAMP column was added to an existing table until all rows are guaranteed to have been materialized (SQL2219, reason code 13, is returned for this error). This can be accomplished with a **LOAD REPLACE** command or with a classic table reorg. This will prevent *false positives*. Tables created with the ROW CHANGE TIMESTAMP column have no restrictions.

Considerations for implicitly hidden columns

A column defined as IMPLICITLY HIDDEN is not part of the result table of a query that specifies * in a SELECT list. However, an implicitly hidden column can be explicitly referenced in a query.

If a column list is not specified on the insert, then the VALUES clause or the SELECT LIST for the insert should not include this column (in general, it must be a generated, defaultable, or nullable column).

For example, an implicitly hidden column can be referenced in the SELECT list, or in a predicate in a query. Additionally, an implicitly hidden column can be explicitly referenced in a CREATE INDEX statement, ALTER TABLE statement, INSERT statement, MERGE statement, or UPDATE statement. An implicitly hidden column can be referenced in a referential constraint. A REFERENCES clause that does not contain a column list refers implicitly to the primary key of the parent

table. It is possible that the primary key of the parent table includes a column defined as implicitly hidden. Such a referential constraint is allowed.

- If the SELECT list of the fullselect of the materialized query definition explicitly refers to an implicitly hidden column, that column will be part of the materialized query table. Otherwise, an implicitly hidden column is not part of a materialized query table that refers to a table containing an implicitly hidden column.
- If the SELECT list of the fullselect of a view definition (CREATE VIEW statement) explicitly refers to an implicitly hidden column, that column will be part of the view, (however the view column is not considered to be hidden). Otherwise, an implicitly hidden column is not part of a view that refers to a table containing an implicitly hidden column.

Considerations for Label Based Access Control (LBAC)

When a column is protected under LBAC, access by a user to that column is determined by the LBAC policies and the security label of the user. This protection, if applied to a row change timestamp column, extends to the reference to that column via both the ROW CHANGE TIMESTAMP and ROW CHANGE TOKEN expressions which are derived from that column.

Therefore when determining the security policies for a table, ensure that the access to the row change timestamp column is available for all users which need to use optimistic locking or time based update detection as appropriate. Note that if there is no row change timestamp column then the ROW CHANGE TOKEN expression cannot be blocked by LBAC. However, if the table is altered to add a row change timestamp column then any LBAC considerations will then apply.

Granularity of row change tokens and false negatives

The RID_BIT() built-in function and the row change token are the only requirements for optimistic locking. However, the schema of the table also affects the behavior of optimistic locking.

For example, a row change timestamp column, defined using either of the following statement clauses, causes the DB2 server to store the time when a row is last changed (or initially inserted). This provides a way to capture the timestamp of the most recent change to a row. This is a timestamp column and it is maintained by the database manager, unless the GENERATED BY DEFAULT clause is used to accept a user-provided input value.

```
GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
GENERATED BY DEFAULT FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
```

Therefore, when an application uses the new ROW CHANGE TOKEN expression on a table, there are two possibilities to consider:

- *The table does not have a row change timestamp column:* A ROW CHANGE TOKEN expression returns a derived BIGINT value that is shared by all rows located on the same page. If one row on a page is updated, the ROW CHANGE TOKEN is changed for all the rows on the same page. This means an update can fail when changes are made to other rows, a property referred to as a false negative.

  Note: Use this mode only if the application can tolerate *false negatives* and does not want to add additional storage to each row for a ROW CHANGE TIMESTAMP column.
- *The table has a row change timestamp column:* A ROW CHANGE TOKEN expression returns a BIGINT value derived from the timestamp value in the

column. In this case, *false negatives* can occur but are more infrequent: If the table is reorganized or redistributed, *false negatives* can occur if the row is moved and an application uses the prior RID_BIT() value.

Time-based update detection

Some applications need to know database updates for certain time ranges, which might be used for replication of data, auditing scenarios, and so on. The ROW CHANGE TIMESTAMP expression provides this information.

```
ROW CHANGE TIMESTAMP FOR table-designator
```

returns a timestamp representing the time when a row was last changed, expressed in local time similar to CURRENT TIMESTAMP. For a row that was updated, this reflects the most recent update to the row. Otherwise, the value corresponds to the original insert of the row.

The value of the ROW CHANGE TIMESTAMP is different from the CURRENT TIMESTAMP in that it is guaranteed unique when assigned by the database per row per database partition. It is a local timestamp approximation of the modification time of each individual row inserted or updated. Since the value is always growing from earlier to later, it can become out of sync with the system clock if:

- The system clock is changed
- The row change timestamp column is GENERATED BY DEFAULT (intended for data propagation only) and a row is provided with an out of sync value.

The prerequisite for using the ROW CHANGE TIMESTAMP expression is that the table must have a row change timestamp column defined using the default precision for the timestamp data type, TIMESTAMP(6) (or TIMESTAMP - the default precision is 6). Every row returns the timestamp of when it was inserted or last updated. There are two methods in which the row change timestamp column can be part of the table:

- The table was created using the FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP clause of the CREATE TABLE command. A ROW CHANGE TIMESTAMP expression returns the value of the column. For this category, the timestamp is precise. The row change timestamp in general when generated by the database is limited by speed of inserts and possible clock manipulations including DST adjustment.
- The table was not created with a row change timestamp column, but one was later added using the FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP clause of the an ALTER TABLE statement. A ROW CHANGE TIMESTAMP expression returns the value of the column. For this category, the old (pre-alter) rows do not contain the actual timestamp until they are first updated or an offline table reorganization is performed.

  Note: The timestamp is an approximate time that the actual update occurred in the database, as of the system clock at the time and taking into account the limitation that no timestamps can be repeated within a database/table partition. In practice this is normally a very accurate representation of the time of the update. The row change timestamp, in general, when generated by the database, is limited by speed of inserts and possible clock manipulations including DST adjustments.

  Rows that were not updated since the ALTER TABLE statement will return the type default value for the column, which is midnight Jan 01, year 1. Only rows that were updated have a unique timestamp. Rows which have the timestamp materialized via an offline table reorganization return a unique timestamp

generated during the reorganization of the table. A **REORG TABLE** operation with the **INPLACE** parameter is not sufficient as it does not materialize schema changes.

In either case, the timestamp of a row might also be updated if a redistribution is performed. If the row is moved from one database partition to another during a redistribution, then a new timestamp must be generated which is guaranteed to be unique at the target.

Time values generated for ROW CHANGE TIMESTAMPs

There are some boundary conditions on the exact values generated for the row change timestamp columns due to the enforcement of unique values per partition.

Whenever the system clock is adjusted into the past for clock correction or for a daylight saving time policy on the DB2 server, it is possible that timestamps will appear to be in the future relative to the current value of the system clock, or the value of the CURRENT TIMESTAMP special register. This occurs when a timestamp was generated before the system clock adjustment, that is, later than the adjusted time, as the timestamps are always generated in an ascending fashion to maintain uniqueness.

When timestamps are generated for columns which were added to the table by a **REORG TABLE** operation or as part of a **LOAD** operation, the timestamps are sequentially generated at some point in the processing of the utility starting from an initial timestamp value. If the utility is able to process rows faster than the timestamp granularity (that is, more than 1 million rows per second), then the values generated for some of the rows can also appear to be in the future relative to the system clock or the CURRENT TIMESTAMP special register.

In each case, when the system clock catches up to the row change timestamp values, there is a close approximation of the time that the row was inserted. Until such time, timestamps are generated in ascending sequence by the finest granularity allowed by the timestamp(6) data type.

RID_BIT() and RID() built-in function

The RID_BIT() and ROW CHANCE TOKEN can be selected for every row in a table. The SELECT can occur at any isolation level that the application requires.

The application can modify the same (unchanged) row with optimistic locking by searching on both:

- The RID_BIT() to directly access (not scan) the target row
- The ROW CHANGE TOKEN to ensure this is the same unchanged row

This update (or delete) can occur at any point after the select, within the same unit of work, or even across connection boundaries; the only requirement is that you obtained the two values listed previously for a given row at some point in time.

Optimistic locking is used in the “WebSphere-Oriented Programming Model”. For example, Microsoft .NET uses this model to process SELECT statements followed by UPDATE or DELETE statements as follows:

- Connect to the database server and SELECT the wanted rows from a table
- Disconnect from the database, or release the row locks so that other applications can read, update, delete, and insert data without any concurrency conflicts due to locks and resources held by the application. Isolation “Uncommited Read” allows higher concurrency AND assuming other applications COMMIT their

update and delete transactions, then this optimistic locking application reads the updated values and the optimistic searched update/delete succeeds.
- Perform some local calculations on the SELECTed row data
- Reconnect to the database server, and search for UPDATE or DELETE on one or more particular targeted rows (and, if the target row changed, handle failed UPDATE or DELETE statements).

Applications using this programming model benefit from the enhanced optimistic locking feature. Note that applications that do not use this programming model are not considered optimistic locking applications, and they continue to work as before.

RID_BIT() and RID() built-in function features

Following are the new features that are implemented for enhanced optimistic locking and for update detection:

RID_BIT(*table-designator* **)**
A new built-in function that returns the Record identifier (RID) of a row as VARCHAR(16) FOR BIT DATA.

Note: DB2 for z/OS implements a built-in function RID with a return type of BIGINT, but that is not large enough for Linux, UNIX, and Windows RIDs. For compatibility, this RID() built-in function returns BIGINT, in addition to RID_BIT().

This RID() built-in function does not work in partitioned database environments, and does not include table version information. Otherwise, it works the same as RID_BIT. Use it only when coding applications that will be ported to z/OS servers. Except where necessary, this topic refers only to RID_BIT.

RID_BIT() built-in function
This built-in function can be used in the SELECT list or predicates statement. In a predicate, for example, `WHERE RID_BIT(tab)=?`, the RID_BIT equals predicate is implemented as a new direct-access method in order to efficiently locate the row. Previously, *optimistic locking with values* was done by adding all the selected column values to the predicates and relying on some unique column combinations to qualify only a single row, with a less efficient access method.

ROW CHANGE TOKEN FOR *table-designator*
A new expression that returns a token as BIGINT. The token represents a relative point in the modification sequence of a row. An application can compare the current ROW CHANGE TOKEN value of a row with the ROW CHANGE TOKEN value that was stored when the row was last fetched to determine whether the row changed.

ROW CHANGE TIMESTAMP column
A GENERATED column with default type of TIMESTAMP which can be defined as either:

```
GENERATED ALWAYS FOR EACH ROW ON UPDATE
AS ROW CHANGE TIMESTAMP
```

or (suggested only for data propagation or unload and reload operations):

```
GENERATED BY DEFAULT FOR EACH ROW ON UPDATE
AS ROW CHANGE TIMESTAMP
```

The data in this column changes every time the row is changed. When this column is defined, the ROW CHANGE TOKEN value is derived from it. Note that when GENERATED ALWAYS is used, the database manager ensures that this value is unique within a database partition or within table partition to ensure that no *false positives* are possible.

To use the first two elements, RID_BIT and ROW CHANGE TOKEN, no other changes are need to the database schema. Note, however, that without the ROW CHANGE TIMESTAMP column, the ROW CHANGE TOKEN is shared by every row on the same page. Updates to any row on the page can cause *false negatives* for other rows stored on the same page. With this column, the ROW CHANGE TOKEN is derived from the timestamp and is not shared with any other rows in the table or database partition. See "Granularity of row change tokens and false negatives" on page 312.

Time-based update detection and RID_BIT(), RID() functions

The ROW CHANGE TIMESTAMP expression returns a timestamp value that represents the time when the row in the table identified by the table designator was last changed. Despite the inter-relation of the RID_BIT() and RID() built-in function and the time-based update detection feature, it is important to note that the usage of ROW CHANGE TOKEN and ROW CHANGE TIMESTAMP expressions are not interchangeable; specifically, that ROW CHANGE TIMESTAMP expression is not part of the optimistic locking usage.

Planning the enablement of optimistic locking

Since the new SQL expressions and attributes for optimistic locking can be used with no DDL changes to the tables involved, you can easily try optimistic locking in your test applications.

Note that without DDL changes, optimistic locking applications might get more *false negatives* than with DDL changes. An application that does get false negatives might not scale well in a production environment because the false negatives might cause too many retries. Therefore, to avoid false negatives, optimistic locking target table(s) should be either:

- Created with a ROW CHANGE TIMESTAMP column
- Altered to contain the ROW CHANGE TIMESTAMP column

If the recommended DDL changes are done, false negatives will be a rare occurrence. The only false negatives will occur due to table level operations such as reorg, not concurrent applications operating on different rows.

In general, the database manager allows false negatives (online or offline reorg, for example) and the presence of a row change timestamp column is sufficient to determine whether page or row level granularity is being used. You can also query the SYSCAT.COLUMNS for a table that has rows with a YES in the ROWCHANGETIMESTAMP column.

A thorough analysis of the application and database might indicate that this DDL is not required, for example, if there is one row per page, or if the update and delete operations are very infrequent and rarely, or never, on the same data page. Such analysis is the exception.

For the update timestamp detection usage, you must make changes to the DDL for the table, and possibly reorganize the table to materialize the values. If there is concern that these changes could have a negative impact on the production

database, you should first prototype the changes in a test environment. For instance, the extra columns can affect the row size limitations and plan selection.

Conditions to be aware of

- You should be aware of conditions relating to the system clock and the granularity of the timestamp values. If a table has a ROW CHANGE TIMESTAMP column, after an insert or update, the new row will have a unique ROW CHANGE TIMESTAMP value in that table on that database partition.
- To ensure uniqueness, the generated timestamp of a row will always increase, regardless if the system clock is adjusted backwards or if the update or insertion of data is happening faster than timestamp granularity. Therefore, the ROW CHANGE TIMESTAMP may be in the future compared with the system time and the CURRENT TIMESTAMP special register. Unless the system clock is gets completely out of sync, or the database manager is inserting or updating at more than one million rows per second, then this should normally be very close to the actual time. In contrast to the CURRENT TIMESTAMP, this value is also generated per row at the time of the update, therefore, it is normally much closer than the CURRENT TIMESTAMP, which is generated once for an entire statement that could take a very long time to complete, depending on the complexity and number of rows affected.

Enabling optimistic locking in applications

There are a number of steps that you must perform in order to enable optimistic locking support in your applications.

Procedure

1. In the initial query, SELECT the row identifier (using the "RID_BIT() and RID() built-in function" on page 314) and ROW CHANGE TOKEN for each row that you need to process.
2. Release the row locks so that other applications can SELECT, INSERT, UPDATE and DELETE from the table.
3. Perform a searched UPDATE or DELETE on the target rows, using the row identifier and ROW CHANGE TOKEN in the search condition, optimistically assuming that the unlocked row has not changed since the original SELECT statement
4. If the row has changed, the UPDATE operation will fail and the application logic must handle the failure. For instance, the application retries the SELECT and UPDATE operations.

What to do next

After running the preceding steps:

- If the number of retries performed by your application seems higher than expected or is desired, then adding a row change timestamp column to your table to ensure that only changes to the row identified by the RID_BIT function will invalidate only the ROW CHANGE TOKEN, and not other activity on the same data page.
- To see rows which have been inserted or updated in a given time range, create or alter the table to contain a row change timestamp column. This column will be maintained by the database manager automatically and can be queried using either the column name or the ROW CHANGE TIMESTAMP expression.
- For row change timestamp columns only, if the column is defined with the IMPLICITLY HIDDEN attribute, then it is not externalized when there is an implicit reference to the columns of the table. However, an implicitly hidden

column can always be referenced explicitly in SQL statements. This can be useful when adding a column to a table can cause existing applications using implicit column lists to fail.

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 `empltabl`. 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.

Creating tables like existing tables

Creating a new source table might be necessary when the characteristics of the target table do not sufficiently match the characteristics of the source when issuing the ALTER TABLE statement with the ATTACH PARTITION clause.

Before creating a new source table, you can attempt to correct the mismatch between the existing source table and the target table.

Before you begin

To create a table, the privileges held by the authorization ID of the statement must include at least one of the following authorities and privileges:

- CREATETAB authority on the database and USE privilege on the table space, as well as one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema
- DBADM authority

About this task

If attempts to correct the mismatch fail, error SQL20408N or SQL20307N is returned.

Procedure

To create a new source table:

1. Use the **db2look** command to produce the CREATE TABLE statement to create a table identical to the target table:

   ```
   db2look -d source_database_name -t source_table_name -e
   ```

2. Remove the partitioning clause from the **db2look** output and change the name of the table created to a new name (for example, "sourceC").
3. Next, load all of the data from the original source table to the newly created source table, sourceC using a **LOAD FROM CURSOR** command:

   ```
   DECLARE mycurs CURSOR FOR SELECT * FROM source
   LOAD FROM mycurs OF CURSOR REPLACE INTO sourceC
   ```

   If this command fails because the original data is incompatible with the definition of table sourceC, you must transform the data in the original table as it is being transferred to sourceC.
4. After the data is successfully copied to sourceC, submit the ALTER TABLE *target* ...ATTACH *sourceC* statement.

Creating tables for staging data

A *staging table* allows incremental maintenance support for deferred materialized query table. The staging table collects changes that must be applied to the materialized query table to synchronize it with the contents of underlying tables. The use of staging tables eliminates the high lock contention caused by immediate maintenance content when an immediate refresh of the materialized query table is requested. Also, the materialized query tables no longer must be entirely regenerated whenever a REFRESH TABLE is performed.

About this task

Materialized query tables are a powerful way to improve response time for complex queries, especially queries that might require some of the following operations:

- Aggregated data over one or more dimensions
- Joins and aggregates data over a group of tables
- Data from a commonly accessed subset of data
- Repartitioned data from a table, or part of a table, in a partitioned database environment

Here are some of the key restrictions regarding staging tables:

1. The query used to define the materialized query table, for which the staging table is created, must be incrementally maintainable; that is, it must adhere to the same rules as a materialized query table with an immediate refresh option.
2. Only a deferred refresh can have a supporting staging table. The query also defines the materialized query table associated with the staging table. The materialized query table must be defined with REFRESH DEFERRED.
3. When refreshing using the staging tables, only a refresh to the current point in time is supported.
4. Partitioned hierarchy tables and partitioned typed tables are not supported. (Partitioned tables are tables where data is partitioned into multiple storage objects based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement.)

An inconsistent, incomplete, or pending state staging table cannot be used to incrementally refresh the associated materialized query table unless some other operations occur. These operations will make the content of the staging table consistent with its associated materialized query table and its underlying tables, and to bring the staging table out of pending. Following a refresh of a materialized query table, the content of its staging table is cleared and the staging table is set to a normal state. A staging table might also be pruned intentionally by using the SET INTEGRITY statement with the appropriate options. Pruning will change the staging table to an inconsistent state. For example, the following statement forces the pruning of a staging table called STAGTAB1:

```
SET INTEGRITY FOR STAGTAB1 PRUNE;
```

When a staging table is created, it is put in a pending state and has an indicator that shows that the table is inconsistent or incomplete with regard to the content of underlying tables and the associated materialized query table. The staging table needs to be brought out of the pending and inconsistent state in order to start collecting the changes from its underlying tables. While in a pending state, any attempts to make modifications to any of the staging table's underlying tables will fail, as will any attempts to refresh the associated materialized query table.

There are several ways a staging table might be brought out of a pending state; for example:

- SET INTEGRITY FOR <staging table name> STAGING IMMEDIATE UNCHECKED
- SET INTEGRITY FOR <staging table name> IMMEDIATE CHECKED

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 24. 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 24. 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 24. 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.

Modifying tables

This section provides topics on how you can modify tables.

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.

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
```

Changing column properties

Use the ALTER TABLE statement to change column properties, such as nullability, LOB options, scope, constraints and compression attributes, data types, and so on.

For complete details, see the ALTER TABLE statement.

Before you begin

To alter a table, you must have at least one of the following privileges on the table to be altered:

- ALTER privilege
- CONTROL privilege
- DBADM authority
- ALTERIN privilege on the schema of the table

You must have DBADM authority to:

- Change the definition of an existing column.
- Edit and test SQL when changing table columns.
- Validate related objects when changing table columns.

Procedure

To change column properties:

Issue the ALTER TABLE statement. For example, from the command line, enter:

```
ALTER TABLE EMPLOYEE
   ALTER COLUMN WORKDEPT
      SET DEFAULT '123'
```

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 285.

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'
  ```

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.

Recovering inoperative summary tables

Summary tables can become *inoperative* as a result of a revoked SELECT privilege on an underlying table.

About this task

The following steps can help you recover an inoperative summary table:

- Determine the statement that was initially used to create the summary table. You can obtain this information from the TEXT column of the SYSCAT.VIEW catalog view.
- Re-create the summary table by using the CREATE SUMMARY TABLE statement with the same summary table name and same definition.
- Use the GRANT statement to re-grant all privileges that were previously granted on the summary table. (Note that all privileges granted on the inoperative summary table are revoked.)

If you do not want to recover an inoperative summary table, you can explicitly drop it with the DROP TABLE statement, or you can create a new summary table with the same name but a different definition.

An inoperative summary table only has entries 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.

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.

Dropping materialized query or staging tables

You cannot alter a materialized query or staging table, but you can drop it. All indexes, primary keys, foreign keys, and check constraints referencing the table are dropped. All views and triggers that reference the table are made inoperative. All packages depending on any object dropped or marked inoperative are invalidated.

About this task

A materialized query table might be explicitly dropped with the DROP TABLE statement, or it might be dropped implicitly if any of the underlying tables are dropped.

A staging table might be explicitly dropped with the DROP TABLE statement, or it might be dropped implicitly when its associated materialized query table is dropped.

Procedure

To drop a materialized query or staging table using the command line, enter:

```
DROP TABLE table_name
```

The following statement drops the materialized query table XT:

```
DROP TABLE XT
```

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 maximum value for the TIMESTAMP(12) data type (default value: `9999-12-30-00.00.00.000000000000`) for any existing rows.

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 25. 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 26. 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 27. 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 28. 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 29. 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 30. 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 31. 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

```
T1: INSERT INTO policy_info
        (policy_id, coverage)
        VALUES ('S777',7000);

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;
```

Transaction XYZ

```
T2: INSERT INTO policy_info
        (policy_id, coverage)
        VALUES ('T888',8000);

T3: 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 32. 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 33. 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 34. 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 35. 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 36. 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 37. 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 38. 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 39. 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

Table 39. Data in the system-period temporal table (policy_info) before the DELETE statement (continued)

policy_id	coverage	sys_start	sys_end	ts_id
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 40. 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 41. 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

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 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 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 42. 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 43. 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 44. 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 45. 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")
- Load data (see "Load" on page 354)
- Online table move (see "ADMIN_MOVE_TABLE procedure" on page 355)
- Quiesce table space (see "QUIESCE TABLESPACES FOR TABLE command" on page 355)
- Replication (see "Replication" on page 355)
- Roll forward (see "Roll forward" on page 355)
- 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 46. 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 47. 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 48. 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 49. 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 50. 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 51. 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

Table 51. Policy A123 updated (continued)

policy_id	coverage	bus_start	bus_end
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 52. 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 53. 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

Table 53. Merged updated data in an application-period temporal table (policy_info) (continued)

policy_id	coverage	bus_start	bus_end
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 54. 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 55. 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

Table 55. Data in the application-period temporal table (policy_info) after the DELETE statement (continued)

policy_id	coverage	bus_start	bus_end
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 56. 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 57. 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

- 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 58. 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 59. 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 60. 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
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 61. 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 62. 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 63. 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

Table 63. Bitemporal table (policy_info) after policy B345 update (continued)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
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 64. 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 65. 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 66. 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 67. 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 68. 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
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 69. 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
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 70. 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 71. 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
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 72. 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

Table 72. History table (hist_policy_info) after DELETE statement (continued)

policy_id	coverage	bus_start	bus_end	sys_start	sys_end	ts_id
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 73. 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

Table 73. Bitemporal table: policy_info (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	2011-02-28-09.10.12.64959200000	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 74. 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
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
```

Scenarios and examples of tables

This section provides scenarios and examples of tables.

Scenarios: Optimistic locking and time-based detection

Three scenarios are provided that show you how to enable and implement optimistic locking in your applications, with and without time-based detection, and with and without implicitly hidden columns.

Scenario: Using optimistic locking in an application program

This scenario demonstrates how optimistic locking is implemented in an application program, covering six different scenarios.

Consider the following sequence of events in an application designed and enabled for optimistic locking:

```
SELECT QUANTITY, row change token FOR STOCK, RID_BIT(STOCK)
 INTO :h_quantity, :h_rct, :h_rid
 FROM STOCK WHERE PARTNUM = 3500
```

In this scenario, the application logic reads each row. Since this application is enabled for optimistic locking as described in "Enabling optimistic locking in applications" on page 317, the select list includes the RID_BIT() value saved in the :h_rid host variable and the row change token value saved in the :h_rct host variable.

With optimistic locking enabled, the application optimistically assumes any rows targeted for update or delete will remain unchanged, even if they are unprotected by locks. To improve database concurrency, the application removes the row lock(s) using one of the following methods:

- Committing the unit of work, in which case the row locks are removed
- Closing the cursor using the WITH RELEASE clause, in which case the row locks are removed
- Using a lower isolation level:
  - CURSOR STABILITY (CS) in which case the row is not locked after the cursor fetches to the next row, or to the end of the result table.
  - UNCOMMITED READ (UR) in which case any uncommitted data has a new (uncommitted) row change token value. If the uncommitted data is rolled back, then the old committed row change token will be a different value.

    Note: Assuming updates are not normally rolled back, using UR allows the most concurrency.
- Disconnecting from the database, thus releasing all DB2 server resources for the application. (.NET applications often use this mode).

The application processes the rows and decides it wants to optimistically update one of them:

```
UPDATE STOCK SET QUANTITY = QUANTITY - 1
 WHERE row change token FOR STOCK = :h_rct AND
 RID_BIT(STOCK) = :h_rid
```

The UPDATE statement updates the row identified in the SELECT statement shown previously.

The searched UPDATE predicate is planned as a direct fetch to the table:

```
RID_BIT(STOCK) = :h_rid
```

Direct fetch is a very efficient access plan, that is simple for the DB2 optimizer to cost. If the RID_BIT() predicate does not find a row, the row was deleted and the update fails with row not found.

Assuming that the RID_BIT() predicate finds a row, the predicate row change token FOR STOCK = :h_rct will find the row if the row change token has not changed. If the row change token has changed since the SELECT, the searched UPDATE fails with row not found.

Table 75 on page 386 lists the possible scenarios that could occur when optimistic locking is enabled.

Table 75. Scenarios that could occur when optimistic locking is enabled

Scenario ID	Action	Result
Scenario 1	There is a row change timestamp column defined on the table and no other application has changed the row.	The update succeeds as the row change token predicate succeeds for the row identified by :h_rid.
Scenario 2	There is a ROW CHANGE TIMESTAMP defined on the table. Another application updates the row after the select and before the update (and commits), updating the row change timestamp column.	The row change token predicate fails comparing the token generated from the timestamp in the row at the time of the select and the token value of the timestamp currently in the row. So the UPDATE statement fails to find a row.
Scenario 3	There is a ROW CHANGE TIMESTAMP defined on the table. Another application updates the row and so the row has a new row change token. This application selects the row at isolation UR and gets the new uncommitted row change token.	This application runs the UPDATE, which will lock wait until the other application releases its row lock. The row change token predicate will succeed if the other application commits the change with the new token, so the UPDATE succeeds. The row change token predicate will fail if the other application rolls back to the old token, so the UPDATE fails to find a row.
Scenario 4	There is no row change timestamp column defined on the table. Another row is updated, deleted or inserted on the same page, after the select and before the update.	The row change token predicate fails comparing the token because the row change token value for all rows on the page has changed, so the UPDATE statement fails to find a row even though our row has not actually changed. This false negative scenario would not result in an UPDATE failure if a row change timestamp column was added.
Scenario 5	The table has been altered to contain a row change timestamp column, and the row returned in the select has not been modified since the time of the alter. Another application updates the row, adding the row change timestamp column to that row in the process with the current timestamp.	The row change token predicate fails comparing the token generated from before with the token value created from the row change timestamp column so the UPDATE statement fails to find a row. Since the row of interest has actually been changed this is not a false negative scenario.
Scenario 6	The table is reorganized after the select and before the update. The row ID identified by :h_rid does not find a row, or contains a row with a different token so the update fails. This is the form of false negative that cannot be avoided even with the existence of a row change timestamp column in the row.	The row itself is not updated by the reorganization but the RID_BIT portion of the predicate cannot identify the original row after the reorganization.

Scenarios: Optimistic locking using implicitly hidden columns

The following scenarios demonstrate how optimistic locking is implemented in an application program using implicitly hidden columns, that is, a column defined with the IMPLICITLY HIDDEN attribute.

For these scenarios, assume that table SALARY_INFO is defined with three columns, and the first column is an implicitly hidden ROW CHANGE TIMESTAMP column whose values are always generated.

Scenario 1:

In the following statement, the implicitly hidden column is explicitly referenced in the column list and a value is provided for it in the VALUES clause:

```
INSERT INTO SALARY_INFO (UPDATE_TIME, LEVEL, SALARY)
  VALUES (DEFAULT, 2, 30000)
```

Scenario 2:

The following INSERT statement uses an implicit column list. An implicit column list does not include implicitly hidden columns, therefore, the VALUES clause only contains values for the other two columns:

```
INSERT INTO SALARY_INFO
  VALUES (2, 30000)
```

In this case, column UPDATE_TIME must be defined to have a default value, and that default value is used for the row that is inserted.

Scenario 3:

In the following statement, the implicitly hidden column is explicitly referenced in the select list and a value for it appears in the result set:

```
SELECT UPDATE_TIME, LEVEL, SALARY FROM SALARY_INFO
  WHERE LEVEL = 2

  UPDATE_TIME        LEVEL    SALARY
  -------------------------- ----------- -----------
  2006-11-28-10.43.27.560841      2    30000
```

Scenario 4:

In the following statement the column list is generated implicitly through use of the * notation, and the implicitly hidden column does not appear in the result set:

```
SELECT * FROM SALARY_INFO
  WHERE LEVEL = 2

  LEVEL    SALARY
  ----------- -----------
       2    30000
```

Scenario 5:

In the following statement, the column list is generated implicitly through use of the * notation, and the implicitly hidden column value also appears by using the ROW CHANGE TIMESTAMP FOR expression:

```
SELECT ROW CHANGE TIMESTAMP FOR SALARY_INFO
  AS ROW_CHANGE_STAMP, SALARY_INFO.*
  FROM SALARY_INFO WHERE LEVEL = 2
```

The result table will be similar to scenario 3 (column UPDATE_TIME will be ROW_CHANGE_STAMP).

Scenario: Time-based update detection

This scenario demonstrates how optimistic locking is implemented in an application program using update detection by timestamp, covering three different scenarios.

In this scenario, the application selects all rows that have changed in the last 30 days.

```
SELECT * FROM TAB WHERE
  ROW CHANGE TIMESTAMP FOR TAB <=
  CURRENT TIMESTAMP AND
  ROW CHANGE TIMESTAMP FOR TAB >=
  CURRENT TIMESTAMP - 30 days;
```

Scenario 1:

No row change timestamp column is defined on the table. Statement fails

with SQL20431N. This SQL expression is only supported for tables with a row change timestamp column defined.

Note: This scenario will work on z/OS.

Scenario 2:

A row change timestamp column was defined when the table was created:

```
CREATE TABLE TAB ( ..., RCT TIMESTAMP NOT NULL
                        GENERATED ALWAYS
                        FOR EACH ROW ON UPDATE AS
                        ROW CHANGE TIMESTAMP)
```

This statement returns all rows inserted or updated in the last 30 days.

Scenario 3:

A row change timestamp column was added to the table using the ALTER TABLE statement at some point in the last 30 days:

```
ALTER TABLE TAB ADD COLUMN RCT TIMESTAMP NOT NULL
                                GENERATED ALWAYS
                                FOR EACH ROW ON UPDATE AS
                                ROW CHANGE TIMESTAMP
```

This statement returns all the rows in the table. Any rows that have not been modified since the ALTER TABLE statement will use the default value of the timestamp of the ALTER TABLE statement itself, and all other rows that have been modified since then will have a unique timestamp.

Chapter 13. 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 38 shows that a duplicate record is prevented from being added to a table when a unique constraint exists for the table:

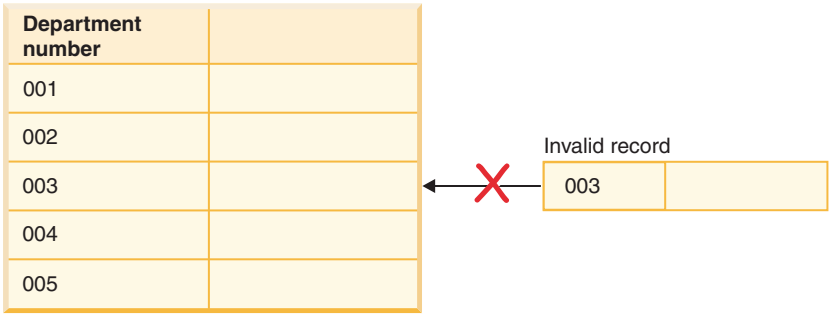


Figure 38. Unique constraints prevent duplicate data

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.

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 39 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:

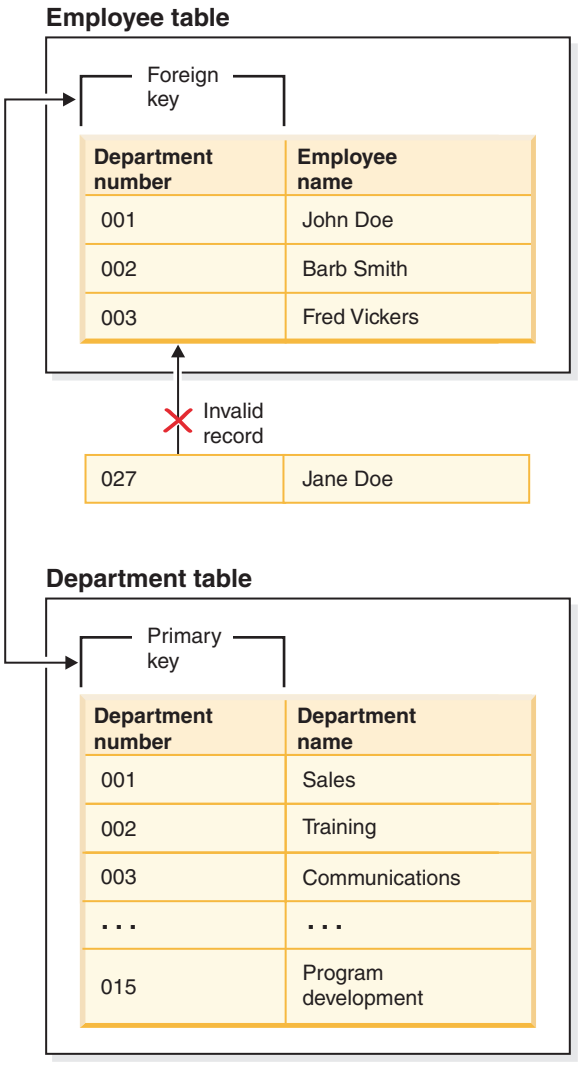


Figure 39. Foreign and primary key constraints

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 76. 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.
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.

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 constraints

When designing and creating constraints, it is a good idea to use a naming convention that properly identifies the different types constraints. This is particularly important for diagnosing errors that might occur.

About this task

You can design the following types of constraints:
- NOT NULL constraints
- Unique constraints
- Primary key constraints
- (Table) Check constraints
- Foreign key (referential) constraints
- Information constraints

Designing unique constraints

Unique constraints ensure that every value in the specified key is unique. A table can have any number of unique constraints, with one unique constraint defined as a primary key.

About this task

You define a unique constraint with the UNIQUE clause in the CREATE TABLE or ALTER TABLE statements. The unique key can consist of more than one column. More than one unique constraint is allowed on a table.

After it is established, the unique constraint is enforced automatically by the database manager when an INSERT or UPDATE statement modifies the data in the table. The unique constraint is enforced through a unique index.

When a unique constraint is defined in an ALTER TABLE statement and an index exists on the same set of columns of that unique key, that index becomes the unique index and is used by the constraint.

You can take any one unique constraint and use it as the primary key. The primary key can be used as the parent key in a referential constraint (along with other unique constraints). You define a primary key with the PRIMARY KEY clause in the CREATE TABLE or ALTER TABLE statement. The primary key can consist of more than one column.

A primary index forces the value of the primary key to be unique. When a table is created with a primary key, the database manager creates a primary index on that key.

Some performance tips for indexes used as unique constraints include:

- When performing an initial load of an empty table with indexes, **LOAD** gives better performance than **IMPORT**. This is true no matter whether you are using the **INSERT** or **REPLACE** modes of **LOAD**.
- When appending a substantial amount of data to an existing table with indexes (using **IMPORT INSERT**, or **LOAD INSERT**), **LOAD** gives slightly better performance than **IMPORT**.
- If you are using the **IMPORT** command for an initial large load of data, create the unique key after the data is imported or loaded. This action avoids maintaining the index when the table is being loaded. It also results in the index using the least amount of storage.
- If you are using the **LOAD** utility in **REPLACE** mode, create the unique key before loading the data. In this case, creation of the index during the load is more efficient than using the CREATE INDEX statement after the load.

Restrictions

- A unique constraint cannot be defined on a subtable.
- There can be only one primary key per table.

Designing primary key constraints

Each table can have one primary key. A primary key 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.

About this task

Because the primary key is used to identify a row in a table, it should be unique and have very few additions or deletions. 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, using the PRIMARY KEY clause. They are also beneficial, because they order the data when data is exported or reorganized.

Primary key constraints are designed like unique constraints, as described in "Designing unique constraints" on page 397. The only difference is that you can have only one primary key constraint per table, whereas, you can have many unique constraints.

Note: You can have primary key constraints based on composite primary keys.

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.

Designing foreign key (referential) constraints

Referential integrity is imposed by adding foreign key (or referential) constraints to table and column definitions, and to create an index on all the foreign key columns. Once the index and foreign key constraints are defined, changes to the data within the tables and columns is checked against the defined constraint. Completion of the requested action depends on the result of the constraint checking.

About this task

Referential constraints are established with the FOREIGN KEY clause, and the REFERENCES clause in the CREATE TABLE or ALTER TABLE statements. There are effects from a referential constraint on a typed table or to a parent table that is a typed table that you should consider before creating a referential constraint.

The identification of foreign keys enforces constraints on the values within the rows of a table or between the rows of two tables. The database manager checks the constraints specified in a table definition and maintains the relationships

accordingly. The goal is to maintain integrity whenever one database object references another, without performance degradation.

For example, primary and foreign keys each have a department number column. For the EMPLOYEE table, the column name is WORKDEPT, and for the DEPARTMENT table, the name is DEPTNO. The relationship between these two tables is defined by the following constraints:

- There is only one department number for each employee in the EMPLOYEE table, and that number exists in the DEPARTMENT table.
- Each row in the EMPLOYEE table is related to no more than one row in the DEPARTMENT table. There is a unique relationship between the tables.
- Each row in the EMPLOYEE table that has a non-null value for WORKDEPT is related to a row in the DEPTNO column of the DEPARTMENT table.
- The DEPARTMENT table is the parent table, and the EMPLOYEE table is the dependent table.

The statement defining the parent table, DEPARTMENT, is:

```
CREATE TABLE DEPARTMENT
       (DEPTNO    CHAR(3)     NOT NULL,
        DEPTNAME  VARCHAR(29) NOT NULL,
        MGRNO     CHAR(6),
        ADMRDEPT  CHAR(3)     NOT NULL,
        LOCATION  CHAR(16),
           PRIMARY KEY (DEPTNO))
    IN RESOURCE
```

The statement defining the dependent table, EMPLOYEE, is:

```
CREATE TABLE EMPLOYEE
       (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
        FIRSTNME  VARCHAR(12) NOT NULL,
        LASTNAME  VARCHAR(15) NOT NULL,
        WORKDEPT  CHAR(3),
        PHONENO   CHAR(4),
        PHOTO     BLOB(10m)   NOT NULL,
           FOREIGN KEY DEPT (WORKDEPT)
           REFERENCES DEPARTMENT ON DELETE NO ACTION)
    IN RESOURCE
```

By specifying the DEPTNO column as the primary key of the DEPARTMENT table and WORKDEPT as the foreign key of the EMPLOYEE table, you are defining a referential constraint on the WORKDEPT values. This constraint enforces referential integrity between the values of the two tables. In this case, any employees that are added to the EMPLOYEE table must have a department number that can be found in the DEPARTMENT table.

The delete rule for the referential constraint in the employee table is NO ACTION, which means that a department cannot be deleted from the DEPARTMENT table if there are any employees in that department.

Although the previous examples use the CREATE TABLE statement to add a referential constraint, the ALTER TABLE statement can also be used.

Another example: The same table definitions are used as those in the previous example. Also, the DEPARTMENT table is created before the EMPLOYEE table. Each department has a manager, and that manager is listed in the EMPLOYEE table. MGRNO of the DEPARTMENT table is actually a foreign key of the EMPLOYEE table. Because of this referential cycle, this constraint poses a slight

problem. You could add a foreign key later. You could also use the CREATE SCHEMA statement to create both the EMPLOYEE and DEPARTMENT tables at the same time.

See also, “Foreign keys in referential constraints” on page 402.

Examples of interaction between triggers and referential constraints

Update operations can cause the interaction of triggers with referential constraints and check constraints.

Figure 40 and the associated description are representative of the processing that is performed for an statement that updates data in the database.

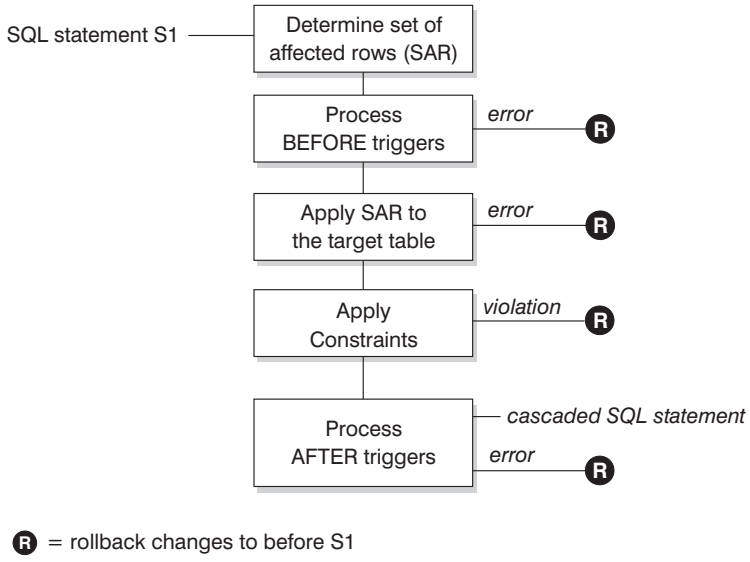


Figure 40. Processing an statement with associated triggers and constraints

Figure 40 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 40.

- 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

Foreign keys in referential constraints

A foreign key references a primary key or a unique key in the same or another table. A foreign key assignment indicates that referential integrity is to be maintained according to the specified referential constraints.

You define a foreign key with the FOREIGN KEY clause in the CREATE TABLE or ALTER TABLE statement. A foreign key makes its table dependent on another table called a parent table. The values in the column or set of columns that make up the foreign key in one table must match the unique key or primary key values of the parent table.

The number of columns in the foreign key must be equal to the number of columns in the corresponding primary or unique constraint (called a parent key) of the parent table. In addition, corresponding parts of the key column definitions must have the same data types and lengths. The foreign key can be assigned a *constraint name*. If you do not assign a name, one is automatically assigned. For ease of use, it is recommended that you assign a *constraint name* and do not use the system-generated name.

The value of a composite foreign key matches the value of a parent key **if** the value of each column of the foreign key is equal to the value of the corresponding column of the parent key. A foreign key containing null values cannot match the values of a parent key, since a parent key by definition can have no null values. However, a null foreign key value is always valid, regardless of the value of any of its non-null parts.

The following rules apply to foreign key definitions:

- A table can have many foreign keys
- A foreign key is nullable if any part is nullable
- A foreign key value is null if any part is null.

When working with foreign keys you can do the following:

- Create a table with zero or more foreign keys.
- Define foreign keys when a table is created or altered.
- Drop foreign keys when a table is altered.

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

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.

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 14. 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 41 on page 414 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 41 shows the relationship between an index and a table.

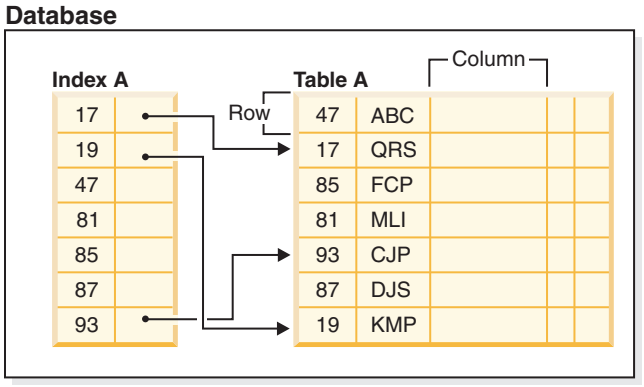


Figure 41. Relationship between an index and a table

Figure 42 illustrates the relationships among some database objects. It also shows that tables, indexes, and long data are stored in table spaces.

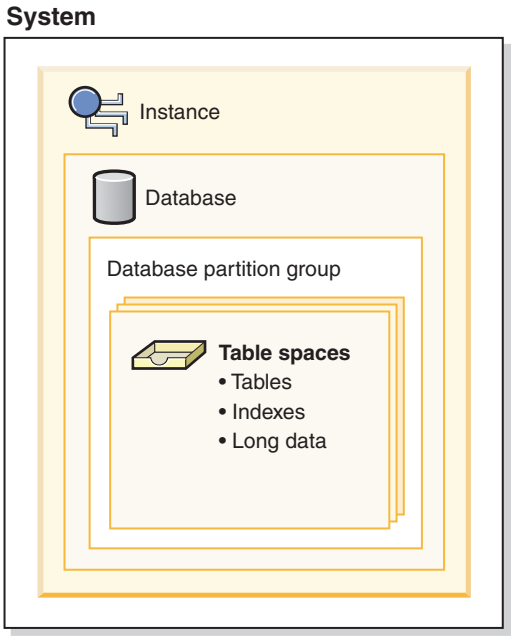


Figure 42. Relationships among selected database objects

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 43 on page 418 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.

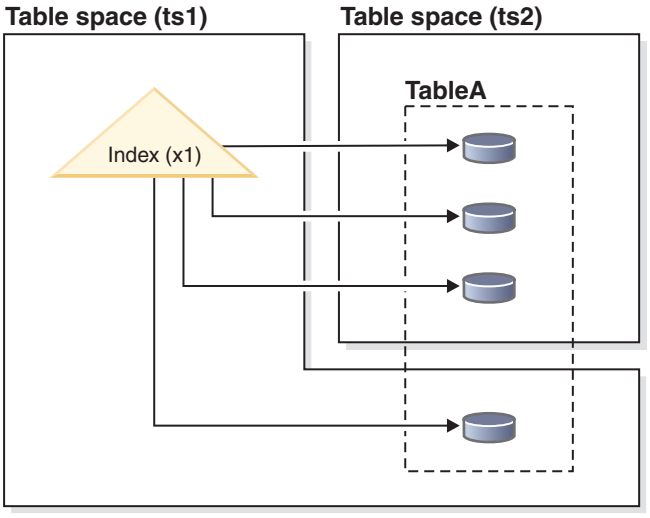


Figure 43. Nonpartitioned index on a partitioned table

Figure 44 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.

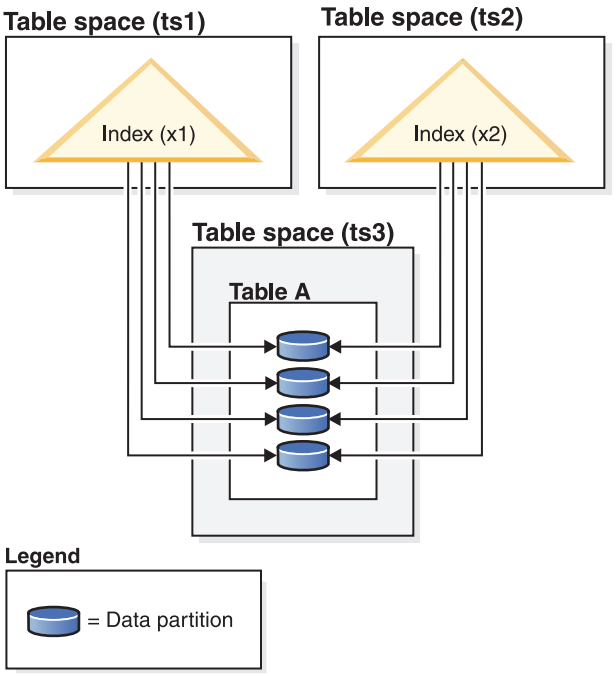


Figure 44. Nonpartitioned indexes on a partitioned table, with indexes in their own table spaces

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 45 shows an example of partitioned indexes.

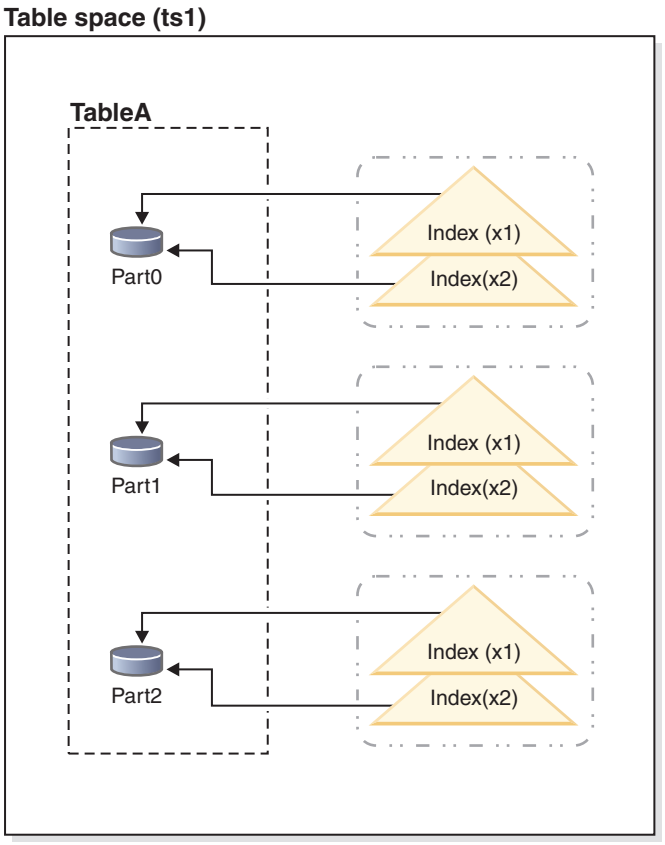


Figure 45. Partitioned indexes that share a table space with data partitions of a table

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 46 on page 421 shows another example of a partitioned index.

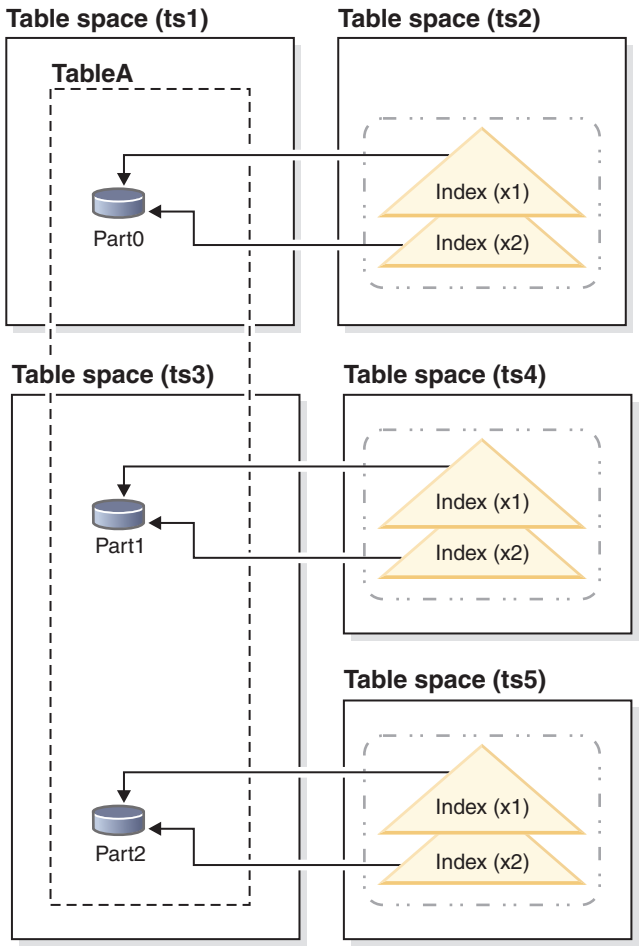


Figure 46. Partitioned indexes with data partitions and index partitions in different table spaces.

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 47 on page 422 diagram shows an example of a partitioned table with both nonpartitioned and partitioned indexes.

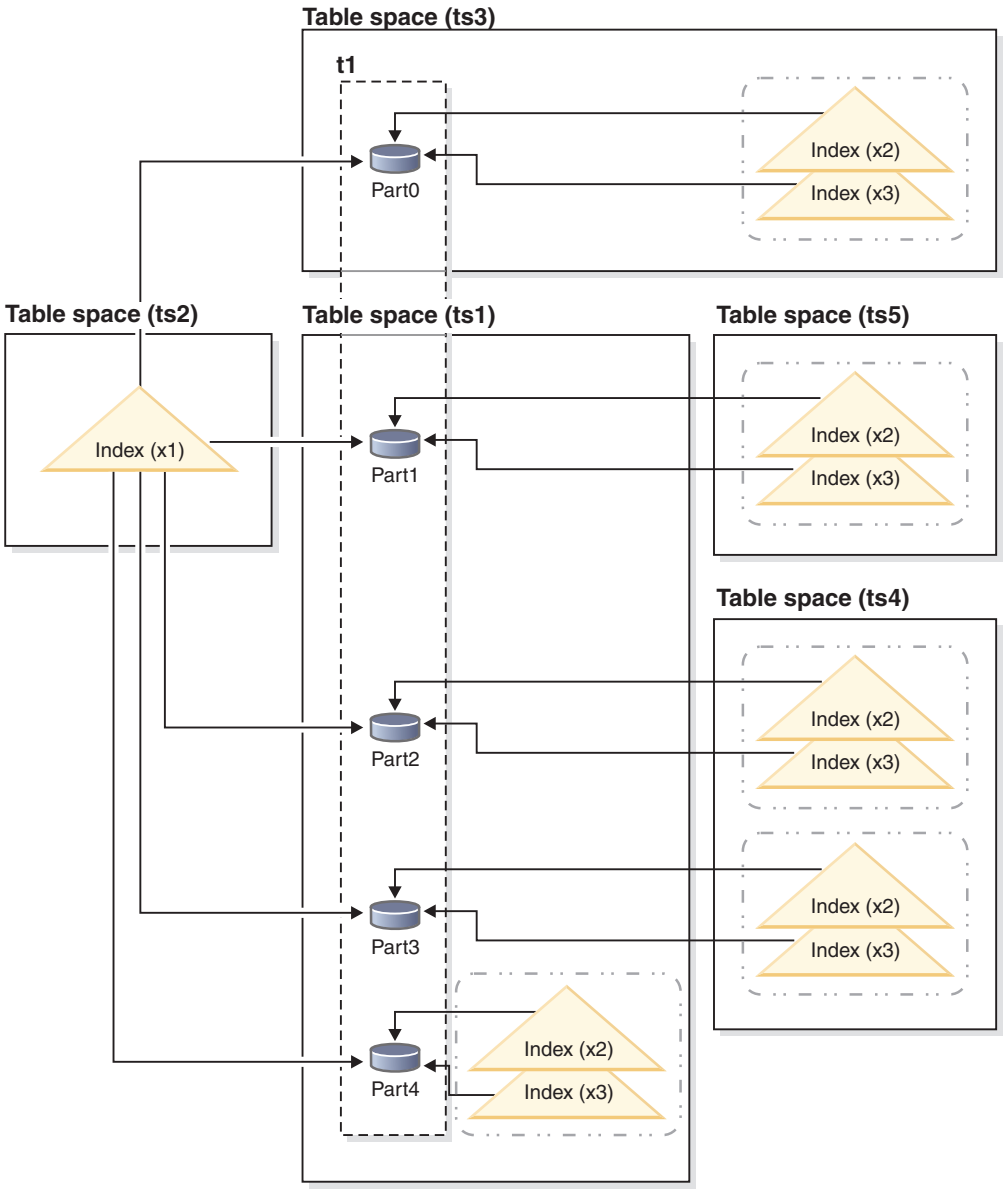


Figure 47. Combination of nonpartitioned and partitioned indexes for a partitioned table

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 77. 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*×2))) × (*D* + 1)) ÷ (*K* + 7 + (*Ds* × *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:

   number of leaf pages = x / (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* ÷ (9 + *minimumKeySize* for non-leaf pages)
   - *K* = *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* ÷ *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:

$T \times$ 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 48 on page 432 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 48. 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 15. 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:

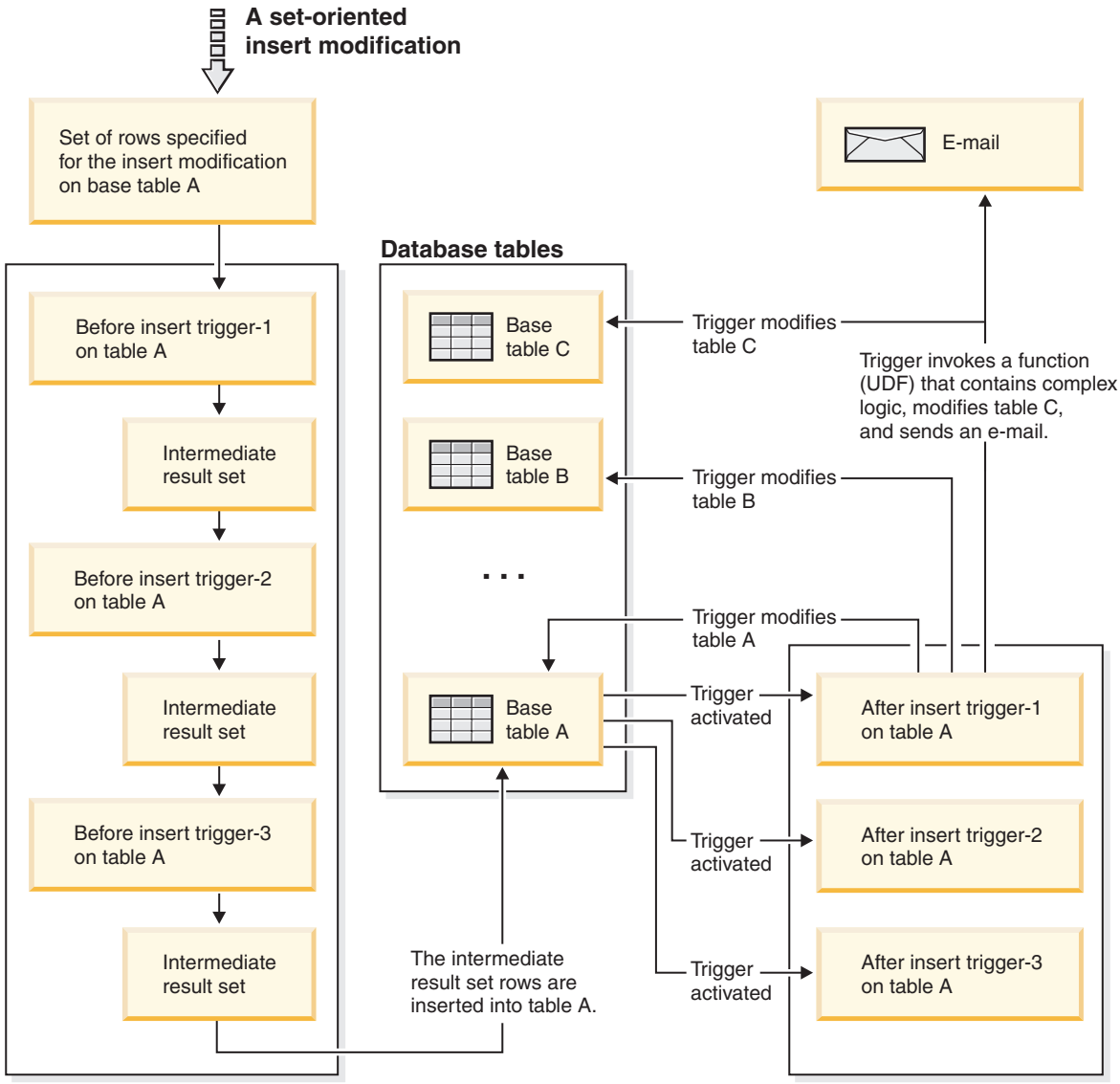


Figure 49. Trigger execution model

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 40 on page 401 and the associated description are representative of the processing that is performed for an statement that updates data in the database.

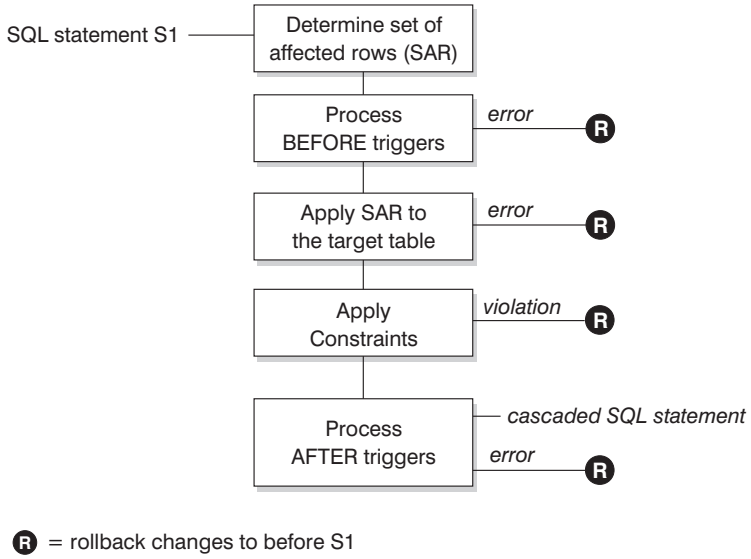


Figure 50. Processing an statement with associated triggers and constraints

Figure 40 on page 401 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 40 on page 401.

- 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 16. 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 464.

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.

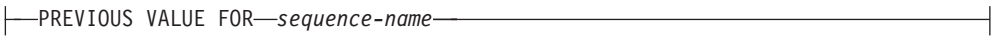
Sequence reference

A sequence reference is an expression which references a sequence defined at the application server.

sequence-reference:

nextval-expression:

prevval-expression:



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 17. 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 51 shows the relationship between tables and views.

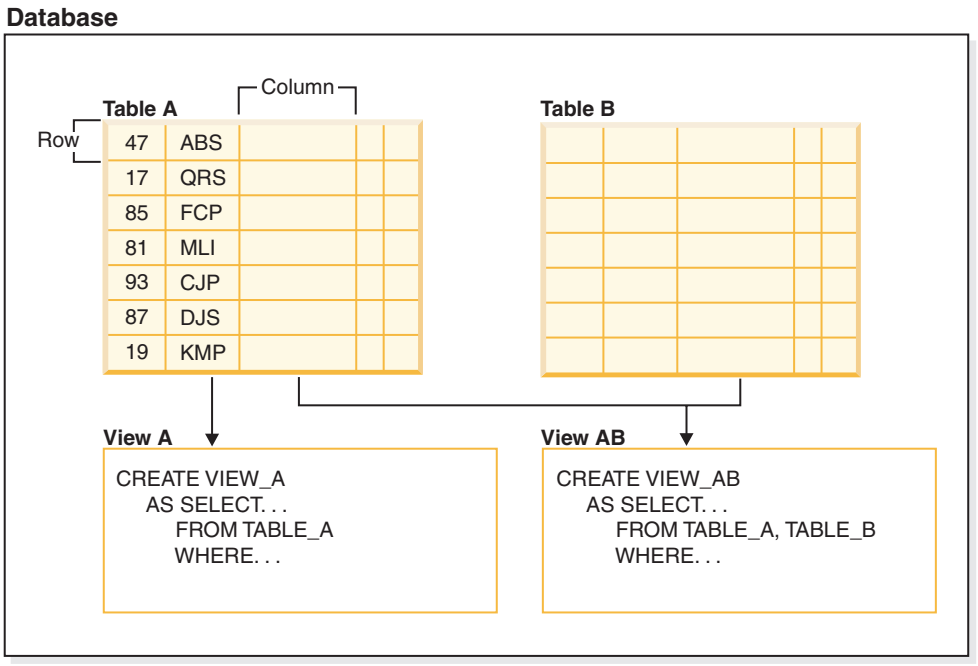


Figure 51. Relationship between tables and views

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 18. Cursors

A *cursor* is used in an application program to select a set of rows and then process that returned data one row at a time. When a SELECT statement in an embedded SQL application returns multiple rows of data, you need a mechanism that makes this returned data or result set available to your application program, one row after another.

A cursor is like a name that is associated with a query. A cursor is created using the DECLARE CURSOR statement which defines the name of the cursor and specifies its associated query. Three additional SQL statements operate on cursors.

OPEN Performs the query which builds the result set and prepares the cursor for retrieval of the first row.

FETCH
Retrieves one row of the result set and assigns the values of that row to target variables. Fetches are usually executed repeatedly until all rows of the result set have been retrieved.

CLOSE
Terminates the cursor and releases any resources that it was using. If needed again, the cursor can be reopened.

You can declare cursors as held or not held, returnable or not returnable, and scrollable or not scrollable.

holdability
A held cursor does not close after a commit operation. A cursor that is not held closes after a commit operation.

returnability
A returnable cursor returns the result set back to the caller of the procedure or client application.

scrollability
A non-scrollable cursor moves sequentially forward through a result set and each row can only be accessed once. A scrollable cursor can be moved randomly through the result set using the FETCH statement. Note that scrollable cursors are supported only in CLI, JDBC, and SQLJ applications.

Chapter 19. 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 78. 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.

Part 4. Reference

Chapter 20. Conforming to naming rules

General naming rules

Rules exist for the naming of all database objects, user names, passwords, groups, files, and paths. Some of these rules are specific to the platform you are working on.

For example, regarding the use of upper and lowercase letters in the names of objects that are visible in the file system (databases, instances, and so on):

- On UNIX platforms, names are case-sensitive. For example, `/data1` is not the same directory as `/DATA1` or `/Data1`
- On Windows platforms, names are not case-sensitive. For example, `\data1` is the same as `\DATA1` and `\Data1`.

Unless otherwise specified, all names can include the following characters:

- The letters A through Z, and a through z, as defined in the basic (7-bit) ASCII character set. When used in identifiers for objects created with SQL statements, lowercase characters "a" through "z" are converted to uppercase unless they are delimited with quotation marks (")
- 0 through 9.
- ! % ( ) { } . - ^ ~ _ (underscore) @, #, $, and space.
- \ (backslash).

Restrictions

- Do not begin names with a number or with the underscore character.
- Do not use SQL reserved words to name tables, views, columns, indexes, or authorization IDs.
- Use only the letters defined in the basic ASCII character set for directory and file names. While your computer's operating system might support different code pages, non-ASCII characters might not work reliably. Using non-ASCII characters can be a particular problem in distributed environment, where different computers might be using different code pages.
- There are other special characters that might work separately depending on your operating system and where you are working with the DB2 database. However, while they might work, there is no guarantee that they will work. It is not recommended that you use these other special characters when naming objects in your database.
- User and group names also must follow the rules imposed by specific operating systems. For example, on Linux and UNIX platforms, characters for user names and group names must be lowercase a through z, 0 through 9, and _ (underscore) for names not starting with 0 through 9.
- Lengths must be less than or equal to the lengths listed in “SQL and XML limits” in the *SQL Reference*.
- **Restrictions on the AUTHID identifier:** In DB2 Version 9.5 and later, you can have a 128-byte authorization ID. However, when the authorization ID is interpreted as an operating system user ID or group name, the operating system naming restrictions apply. For example, the Linux and UNIX operating systems can contain up to 8 characters and the Windows operating systems can contain up to 30 characters for user IDs and group names. Therefore, while you can

grant a 128-byte authorization ID, you cannot connect as a user that has that authorization ID. If you write your own security plug-in, you can use the extended sizes for the authorization ID. For example, you can give your security plug-in a 30-byte user ID and it returns a 128-byte authorization ID during authentication that you can connect to.

You also must consider object naming rules, naming rules in an multicultural support environment, and naming rules in a Unicode environment.

DB2 object naming rules

All objects follow the general gaming rules. In addition, some objects have additional restrictions shown in the accompanying tables.

Table 79. Database, database alias and instance naming rules

Objects	Guidelines
• Databases • Database aliases • Instances	• Database names must be unique within the location in which they are cataloged. On Linux and UNIX implementations, this location is a directory path, whereas on Windows implementations, it is a logical disk. • Database alias names must be unique within the system database directory. When a new database is created, the alias defaults to the database name. As a result, you cannot create a database using a name that exists as a database alias, even if there is no database with that name. • Database, database alias and instance name lengths must be less than or equal to 8 bytes. • On Windows, no instance can have the same name as a service name. **Note:** To avoid potential problems, do not use the special characters @, #, and $ in a database name if you intend to use the database in a communications environment. Also, because these characters are not common to all keyboards, do not use them if you plan to use the database in another language.

Table 80. Database object naming rules

Objects	Guidelines
• Aliases • Audit policies • Buffer pools • Columns • Event monitors • Indexes • Methods • Nodegroups • Packages • Package versions • Roles • Schemas • Stored procedures • Tables • Table spaces • Triggers • Trusted contexts • UDFs • UDTs • Views	• Lengths for identifiers for these objects must be less than or equal to the lengths listed in "SQL and XML limits" in the *SQL Reference*. Object names can also include: – Valid accented characters (such as ö) – Multibyte characters, except multibyte spaces (for multibyte environments) • Package names and package versions can also include periods (.), hyphens (-), and colons (:). For more information, see "Identifiers" in the *SQL Reference*.

Table 81. Federated database object naming rules

Objects	Guidelines
• Function mappings • Index specifications • Nicknames • Servers • Type mappings • User mappings • Wrappers	Lengths for these objects must be less than or equal to the lengths listed in "SQL and XML limits" in the *SQL Reference*. Names for federated database objects can also include: • Valid accented letters (such as ö) • Multibyte characters, except multibyte spaces (for multibyte environments)

Delimited identifiers and object names

Keywords can be used. If a keyword is used in a context where it could also be interpreted as an SQL keyword, it must be specified as a delimited identifier.

Using delimited identifiers, it is possible to create an object that violates these naming rules; however, subsequent use of the object could result in errors. For example, if you create a column with a + or - sign included in the name and you subsequently use that column in an index, you will experience problems when you attempt to reorganize the table.

Additional schema names information

- User-defined types (UDTs) cannot have schema names longer than the lengths 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, SYSPUBLIC.
- 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 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.

Delimited identifiers and object names

Keywords can be used. If a keyword is used in a context where it could also be interpreted as an SQL keyword, it must be specified as a delimited identifier.

Using delimited identifiers, it is possible to create an object that violates these naming rules; however, subsequent use of the object could result in errors. For example, if you create a column with a + or - sign included in the name and you subsequently use that column in an index, you will experience problems when you attempt to reorganize the table.

User, user ID and group naming rules

User, user ID and group names must follow naming guidelines.

Table 82. User, user ID and group naming rules

Objects	Guidelines
• Group names • User names • User IDs	• Group names must be less than or equal to the group name length listed in "SQL and XML limits" in the *SQL Reference*. • User IDs on Linux and UNIX operating systems can contain up to 8 characters. • User names on Windows can contain up to 30 characters. • When not using Client authentication, non-Windows 32-bit clients connecting to Windows with user names longer than the user name length listed in "SQL and XML limits" in the *SQL Reference* are supported when the user name and password are specified explicitly. • Names and IDs cannot: – Be USERS, ADMINS, GUESTS, PUBLIC, LOCAL or any SQL reserved word – Begin with IBM, SQL or SYS.

Note:

1. Some operating systems allow case sensitive user IDs and passwords. You should check your operating system documentation to see if this is the case.
2. The authorization ID returned from a successful CONNECT or ATTACH is truncated to the authorization name length listed in "SQL and XML limits" in

the *SQL Reference*. An ellipsis (...) is appended to the authorization ID and the SQLWARN fields contain warnings to indicate truncation.

3. Trailing blanks from user IDs and passwords are removed.
4. **Restrictions on the AUTHID identifier:** In DB2 Version 9.5 and later, you can have a 128-byte authorization ID. However, when the authorization ID is interpreted as an operating system user ID or group name, the operating system naming restrictions apply. For example, the Linux and UNIX operating systems can contain up to 8 characters and the Windows operating systems can contain up to 30 characters for user IDs and group names. Therefore, while you can grant a 128-byte authorization ID, you cannot connect as a user that has that authorization ID. If you write your own security plug-in, you can use the extended sizes for the authorization ID. For example, you can give your security plug-in a 30-byte user ID and it returns a 128-byte authorization ID during authentication that you can connect to.

Naming rules in a multiple national language environment

The basic character set that can be used in database names consists of the single-byte uppercase and lowercase Latin letters (A...Z, a...z), the Arabic numerals (0...9) and the underscore character (_).

This list is augmented with three special characters (#, @, and $) to provide compatibility with host database products. Use special characters #, @, and $ with care in a multiple national language environment because they are not included in the multiple national language host (EBCDIC) invariant character set. Characters from the extended character set can also be used, depending on the code page that is being used. If you are using the database in a multiple code page environment, you must ensure that all code pages support any elements from the extended character set you plan to use.

When naming database objects (such as tables and views), program labels, host variables, cursors, and elements from the extended character set (for example, letters with diacritical marks) can also be used. Precisely which characters are available depends on the code page in use.

Extended Character Set Definition for DBCS Identifiers: In DBCS environments, the extended character set consists of all the characters in the basic character set, plus the following:

- All double-byte characters in each DBCS code page, except the double-byte space, are valid letters.
- The double-byte space is a special character.
- The single-byte characters available in each mixed code page are assigned to various categories as follows:

Category	Valid Code Points within each Mixed Code Page
Digits	x30-39
Letters	x23-24, x40-5A, x61-7A, xA6-DF (A6-DF for code pages 932 and 942 only)
Special Characters	All other valid single-byte character code points

Naming rules in a Unicode environment

In a Unicode database, all identifiers are in multibyte UTF-8. Therefore, it is possible to use any UCS-2 character in identifiers where the use of a character in the extended character set (for example, an accented character, or a multibyte character) is allowed by the DB2 database system.

Clients can enter any character that is supported by their environment, and all the characters in the identifiers will be converted to UTF-8 by the database manager. Two points must be taken into account when specifying national language characters in identifiers for a Unicode database:

- Each non-ASCII character requires two to four bytes. Therefore, an *n*-byte identifier can only hold somewhere between $n/4$ and n characters, depending on the ratio of ASCII to non-ASCII characters. If you have only one or two non-ASCII (for example, accented) characters, the limit is closer to n characters, whereas for an identifier that is completely non-ASCII (for example, in Japanese), only $n/4$ to $n/3$ characters can be used.
- If identifiers are to be entered from different client environments, they should be defined using the common subset of characters available to those clients. For example, if a Unicode database is to be accessed from Latin-1, Arabic, and Japanese environments, all identifiers should realistically be limited to ASCII.

Chapter 21. Lightweight Directory Access Protocol (LDAP)

Lightweight Directory Access Protocol (LDAP) is an industry standard access method to directory services. A directory service is a repository of resource information about multiple systems and services within a distributed environment; and it provides client and server access to these resources.

Each database server instance publishes its existence to an LDAP server and provides database information to the LDAP directory when the databases are created. When a client connects to a database, the catalog information for the server can be retrieved from the LDAP directory. Each client is no longer required to store catalog information locally on each machine. Client applications search the LDAP directory for information required to connect to the database.

A caching mechanism exists so that the client only needs to search the LDAP directory server once. After the information is retrieved from the LDAP directory server, it is stored or cached on the local computer based on the values of the **dir_cache** database manager configuration parameter and the **DB2LDAPCACHE** registry variable. The **dir_cache** database manager configuration parameter is used to store database, node, and DCS directory files in a memory cache. The directory cache is used by an application until the application closes. The **DB2LDAPCACHE** registry variable is used to store database, node, and DCS directory files in a local disk cache.

- If **DB2LDAPCACHE**=NO and **dir_cache**=NO, then always read the information from LDAP.
- If **DB2LDAPCACHE**=NO and **dir_cache**=YES, then read the information from LDAP once and insert it into the DB2 cache.
- If **DB2LDAPCACHE**=YES or is not set, then read the information from LDAP once and cache it into the local database, node, and DCS directories.

Note: The **DB2LDAPCACHE** registry variable is only applicable to the database and node directories.

Security considerations in an LDAP environment

Before accessing information in the LDAP directory, an application or user is authenticated by the LDAP server. The authentication process is called *binding* to the LDAP server. It is important to apply access control on the information stored in the LDAP directory to prevent anonymous users from adding, deleting, or modifying the information.

Access control is inherited by default and can be applied at the container level. When a new object is created, it inherits the same security attribute as the parent object. An administration tool available for the LDAP server can be used to define access control for the container object.

By default, access control is defined as follows:

- For database and node entries in LDAP, everyone (or any anonymous user) has read access. Only the Directory Administrator and the owner or creator of the object has read/write access.

- For user profiles, the profile owner and the Directory Administrator have read/write access. One user cannot access the profile of another user if that user does not have Directory Administrator authority.

Note: The authorization check is always performed by the LDAP server and not by DB2. The LDAP authorization check is not related to DB2 authorization. An account or authorization ID that has SYSADM authority might not have access to the LDAP directory.

When running the LDAP commands or APIs, if the bind Distinguished Name (bindDN) and password are not specified, DB2 binds to the LDAP server using the default credentials which might not have sufficient authority to perform the requested commands and an error will be returned.

You can explicitly specify the user's bindDN and password using the USER and PASSWORD clauses for the DB2 commands or APIs.

LDAP object classes and attributes used by DB2

The following tables describe the object classes that are used by the DB2 database manager:

Table 83. cimManagedElement

Class	**cimManagedElement**
Active Directory LDAP Display Name	Not applicable
Active Directory Common Name (cn)	Not applicable
Description	Provides a base class of many of the system management object classes in the IBM Schema
SubClassOf	top
Required Attribute(s)	
Optional Attribute(s)	description
Type	abstract
OID (Object Identifier)	1.3.18.0.2.6.132
GUID (Global Unique Identifier)	b3afd63f-5c5b-11d3-b818-002035559151

Table 84. cimSetting

Class	**cimSetting**
Active Directory LDAP Display Name	Not applicable
Active Directory Common Name (cn)	Not applicable
Description	Provides a base class for configuration and settings in the IBM Schema
SubClassOf	cimManagedElement
Required Attribute(s)	
Optional Attribute(s)	settingID
Type	abstract
OID (object identifier)	1.3.18.0.2.6.131
GUID (Global Unique Identifier)	b3afd64d-5c5b-11d3-b818-002035559151

Table 85. eProperty

Class	**eProperty**
Active Directory LDAP Display Name	ibm-eProperty
Active Directory Common Name (cn)	ibm-eProperty
Description	Used to specify any application specific settings for user preference properties
SubClassOf	cimSetting
Required Attribute(s)	
Optional Attribute(s)	propertyType cisPropertyType cisProperty cesPropertyType cesProperty binPropertyType binProperty
Type	structural
OID (object identifier)	1.3.18.0.2.6.90
GUID (Global Unique Identifier)	b3afd69c-5c5b-11d3-b818-002035559151

Table 86. DB2Node

Class	**DB2Node**
Active Directory LDAP Display Name	ibm-db2Node
Active Directory Common Name (cn)	ibm-db2Node
Description	Represents a DB2 Server
SubClassOf	eSap / ServiceConnectionPoint
Required Attribute(s)	db2nodeName
Optional Attribute(s)	db2nodeAlias db2instanceName db2Type host / dNSHostName (see Note 2) protocolInformation/ServiceBindingInformation
Type	structural
OID (object identifier)	1.3.18.0.2.6.116
GUID (Global Unique Identifier)	b3afd65a-5c5b-11d3-b818-002035559151

Table 86. DB2Node (continued)

Class	DB2Node
Special Notes	1. The *DB2Node* class is derived from *eSap* object class under IBM Tivoli® Directory Server and from *ServiceConnectionPoint* object class under Microsoft Active Directory. 2. The *host* is used under the IBM Tivoli Directory Server environment. The *dNSHostName* attribute is used under Microsoft Active Directory. 3. The *protocolInformation* is only used under the IBM Tivoli Directory Server environment. For Microsoft Active Directory, the attribute *ServiceBindingInformation*, inherited from the ServiceConnectionPoint class, is used to contain the protocol information.

The *protocolInformation* (in IBM Tivoli Directory Server) or *ServiceBindingInformation* (in Microsoft Active Directory) attribute in the *DB2Node* object contains the communication protocol information to bind the DB2 database server. It consists of tokens that describe the network protocol supported. Each token is separated by a semicolon. There is no space between the tokens. An asterisk (*) can be used to specify an optional parameter.

The tokens for TCP/IP are:

- "TCPIP"
- Server hostname or IP address
- Service name (svcename) or port number (for example 50000)
- (Optional) security ("NONE" or "SOCKS")

The tokens for Named Pipe are:

- "NPIPE"
- Computer name of the server
- Instance name of the server

Table 87. DB2Database

Class	DB2Database
Active Directory LDAP Display Name	ibm-db2Database
Active Directory Common Name (cn)	ibm-db2Database
Description	Represents a DB2 database
SubClassOf	top
Required Attribute(s)	db2databaseName db2nodePtr

Table 87. DB2Database (continued)

Class	**DB2Database**
Optional Attribute(s)	db2databaseAlias db2additionalParameter db2ARLibrary db2authenticationLocation db2gwPtr db2databaseRelease DCEPrincipalName db2altgwPtr db2altnodePtr
Type	structural
OID (object identifier)	1.3.18.0.2.6.117
GUID (Global Unique Identifier)	b3afd659-5c5b-11d3-b818-002035559151

Table 88. db2additionalParameters

Attribute	**db2additionalParameters**
Active Directory LDAP Display Name	ibm-db2AdditionalParameters
Active Directory Common Name (cn)	ibm-db2AdditionalParameters
Description	Contains any additional parameters used when connecting to the host database server
Syntax	Case Ignore String
Maximum Length	1024
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.426
GUID (Global Unique Identifier)	b3afd315-5c5b-11d3-b818-002035559151

Table 89. db2authenticationLocation

Attribute	**db2authenticationLocation**
Active Directory LDAP Display Name	ibm-db2AuthenticationLocation
Active Directory Common Name (cn)	ibm-db2AuthenticationLocation
Description	Specifies where authentication takes place
Syntax	Case Ignore String
Maximum Length	64
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.425
GUID (Global Unique Identifier)	b3afd317-5c5b-11d3-b818-002035559151
Notes	Valid values are: CLIENT, SERVER, DCS, DCE, KERBEROS, SVRENCRYPT, or DCSENCRYPT

Table 90. db2ARLibrary

Attribute	**db2ARLibrary**
Active Directory LDAP Display Name	ibm-db2ARLibrary
Active Directory Common Name (cn)	ibm-db2ARLibrary
Description	Name of the Application Requestor library
Syntax	Case Ignore String
Maximum Length	256
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.427
GUID (Global Unique Identifier)	b3afd316-5c5b-11d3-b818-002035559151

Table 91. db2databaseAlias

Attribute	**db2databaseAlias**
Active Directory LDAP Display Name	ibm-db2DatabaseAlias
Active Directory Common Name (cn)	ibm-db2DatabaseAlias
Description	Database alias name(s)
Syntax	Case Ignore String
Maximum Length	1024
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.422
GUID (Global Unique Identifier)	b3afd318-5c5b-11d3-b818-002035559151

Table 92. db2databaseName

Attribute	**db2databaseName**
Active Directory LDAP Display Name	ibm-db2DatabaseName
Active Directory Common Name (cn)	ibm-db2DatabaseName
Description	Database name
Syntax	Case Ignore String
Maximum Length	1024
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.421
GUID (Global Unique Identifier)	b3afd319-5c5b-11d3-b818-002035559151

Table 93. db2databaseRelease

Attribute	**db2databaseRelease**
Active Directory LDAP Display Name	ibm-db2DatabaseRelease
Active Directory Common Name (cn)	ibm-db2DatabaseRelease
Description	Database release number
Syntax	Case Ignore String
Maximum Length	64
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.429

Table 93. db2databaseRelease (continued)

Attribute	db2databaseRelease
GUID (Global Unique Identifier)	b3afd31a-5c5b-11d3-b818-002035559151

Table 94. db2nodeAlias

Attribute	db2nodeAlias
Active Directory LDAP Display Name	ibm-db2NodeAlias
Active Directory Common Name (cn)	ibm-db2NodeAlias
Description	Node alias name(s)
Syntax	Case Ignore String
Maximum Length	1024
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.420
GUID (Global Unique Identifier)	b3afd31d-5c5b-11d3-b818-002035559151

Table 95. db2nodeName

Attribute	db2nodeName
Active Directory LDAP Display Name	ibm-db2NodeName
Active Directory Common Name (cn)	ibm-db2NodeName
Description	Node name
Syntax	Case Ignore String
Maximum Length	64
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.419
GUID (Global Unique Identifier)	b3afd31e-5c5b-11d3-b818-002035559151

Table 96. db2nodePtr

Attribute	db2nodePtr
Active Directory LDAP Display Name	ibm-db2NodePtr
Active Directory Common Name (cn)	ibm-db2NodePtr
Description	Pointer to the Node (DB2Node) object that represents the database server which owns the database
Syntax	Distinguished Name
Maximum Length	1000
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.423
GUID (Global Unique Identifier)	b3afd31f-5c5b-11d3-b818-002035559151
Special Notes	This relationship allows the client to retrieve protocol communication information to connect to the database

Table 97. db2altnodePtr

Attribute	db2altnodePtr
Active Directory LDAP Display Name	ibm-db2AltNodePtr

Table 97. db2altnodePtr (continued)

Attribute	db2altnodePtr
Active Directory Common Name (cn)	ibm-db2AltNodePtr
Description	Pointer to the Node (DB2Node) object that represents the alternate database server
Syntax	Distinguished Name
Maximum Length	1000
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.3093
GUID (Global Unique Identifier)	5694e266-2059-4e32-971e-0778909e0e72

Table 98. db2gwPtr

Attribute	db2gwPtr
Active Directory LDAP Display Name	ibm-db2GwPtr
Active Directory Common Name (cn)	ibm-db2GwPtr
Description	Pointer to the Node object that represents the gateway server and from which the database can be accessed
Syntax	Distinguished Name
Maximum Length	1000
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.424
GUID (Global Unique Identifier)	b3afd31b-5c5b-11d3-b818-002035559151

Table 99. db2altgwPtr

Attribute	db2altgwPtr
Active Directory LDAP Display Name	ibm-db2AltGwPtr
Active Directory Common Name (cn)	ibm-db2AltGwPtr
Description	Pointer to the Node object that represents the alternate gateway server
Syntax	Distinguished Name
Maximum Length	1000
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.3092
GUID (Global Unique Identifier)	70ab425d-65cc-4d7f-91d8-084888b3a6db

Table 100. db2instanceName

Attribute	db2instanceName
Active Directory LDAP Display Name	ibm-db2InstanceName
Active Directory Common Name (cn)	ibm-db2InstanceName
Description	The name of the database server instance
Syntax	Case Ignore String
Maximum Length	256
Multi-Valued	Single-valued

Table 100. db2instanceName (continued)

Attribute	**db2instanceName**
OID (object identifier)	1.3.18.0.2.4.428
GUID (Global Unique Identifier)	b3afd31c-5c5b-11d3-b818-002035559151

Table 101. db2Type

Attribute	**db2Type**
Active Directory LDAP Display Name	ibm-db2Type
Active Directory Common Name (cn)	ibm-db2Type
Description	Type of the database server
Syntax	Case Ignore String
Maximum Length	64
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.418
GUID (Global Unique Identifier)	b3afd320-5c5b-11d3-b818-002035559151
Notes	Valid types for database server are: SERVER, MPP, and DCS

Table 102. DCEPrincipalName

Attribute	**DCEPrincipalName**
Active Directory LDAP Display Name	ibm-DCEPrincipalName
Active Directory Common Name (cn)	ibm-DCEPrincipalName
Description	DCE principal name
Syntax	Case Ignore String
Maximum Length	2048
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.443
GUID (Global Unique Identifier)	b3afd32d-5c5b-11d3-b818-002035559151

Table 103. cesProperty

Attribute	**cesProperty**
Active Directory LDAP Display Name	ibm-cesProperty
Active Directory Common Name (cn)	ibm-cesProperty
Description	Values of this attribute can be used to provide application-specific preference configuration parameters. For example, a value can contain XML-formatted data. All values of this attribute must be homogeneous in the cesPropertyType attribute value.
Syntax	Case Exact String
Maximum Length	32700
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.307
GUID (Global Unique Identifier)	b3afd2d5-5c5b-11d3-b818-002035559151

Table 104. cesPropertyType

Attribute	cesPropertyType
Active Directory LDAP Display Name	ibm-cesPropertyType
Active Directory Common Name (cn)	ibm-cesPropertyType
Description	Values of this attribute can be used to describe the syntax, semantics, or other characteristics of all of the values of the cesProperty attribute. For example, a value of "XML" might be used to indicate that all the values of the cesProperty attribute are encoded as XML syntax.
Syntax	Case Ignore String
Maximum Length	128
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.308
GUID (Global Unique Identifier)	b3afd2d6-5c5b-11d3-b818-002035559151

Table 105. cisProperty

Attribute	cisProperty
Active Directory LDAP Display Name	ibm-cisProperty
Active Directory Common Name (cn)	ibm-cisProperty
Description	Values of this attribute can be used to provide application-specific preference configuration parameters. For example, a value can contain an INI file. All values of this attribute must be homogeneous in their cisPropertyType attribute value.
Syntax	Case Ignore String
Maximum Length	32700
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.309
GUID (Global Unique Identifier)	b3afd2e0-5c5b-11d3-b818-002035559151

Table 106. cisPropertyType

Attribute	cisPropertyType
Active Directory LDAP Display Name	ibm-cisPropertyType
Active Directory Common Name (cn)	ibm-cisPropertyType
Description	Values of this attribute can be used to describe the syntax, semantics, or other characteristics of all of the values of the cisProperty attribute. For example, a value of "INI File" might be used to indicate that all the values of the cisProperty attribute are INI files.
Syntax	Case Ignore String
Maximum Length	128
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.310
GUID (Global Unique Identifier)	b3afd2e1-5c5b-11d3-b818-002035559151

Table 107. binProperty

Attribute	**binProperty**
Active Directory LDAP Display Name	ibm-binProperty
Active Directory Common Name (cn)	ibm-binProperty
Description	Values of this attribute can be used to provide application-specific preference configuration parameters. For example, a value can contain a set of binary-encoded Lotus® 123 properties. All values of this attribute must be homogeneous in their binPropertyType attribute values.
Syntax	binary
Maximum Length	250000
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.305
GUID (Global Unique Identifier)	b3afd2ba-5c5b-11d3-b818-002035559151

Table 108. binPropertyType

Attribute	**binPropertyType**
Active Directory LDAP Display Name	ibm-binPropertyType
Active Directory Common Name (cn)	ibm-binPropertyType
Description	Values of this attribute can be used to describe the syntax, semantics, or other characteristics of all of the values of the binProperty attribute. For example, a value of “Lotus 123” might be used to indicate that all the values of the binProperty attribute are binary-encoded Lotus 123 properties.
Syntax	Case Ignore String
Maximum Length	128
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.306
GUID (Global Unique Identifier)	b3afd2bb-5c5b-11d3-b818-002035559151

Table 109. PropertyType

Attribute	**PropertyType**
Active Directory LDAP Display Name	ibm-propertyType
Active Directory Common Name (cn)	ibm-propertyType
Description	Values of this attribute describe the semantic characteristics of the eProperty object
Syntax	Case Ignore String
Maximum Length	128
Multi-Valued	Multi-valued
OID (object identifier)	1.3.18.0.2.4.320
GUID (Global Unique Identifier)	b3afd4ed-5c5b-11d3-b818-002035559151

Table 110. settingID

Attribute	settingID
Active Directory LDAP Display Name	Not applicable
Active Directory Common Name (cn)	Not applicable
Description	A naming attribute that can be used to identify the cimSetting derived object entries such as eProperty
Syntax	Case Ignore String
Maximum Length	256
Multi-Valued	Single-valued
OID (object identifier)	1.3.18.0.2.4.325
GUID (Global Unique Identifier)	b3afd596-5c5b-11d3-b818-002035559151

Extending the LDAP directory schema with DB2 object classes and attributes

The LDAP Directory Schema defines object classes and attributes for the information stored in the LDAP directory entries. An object class consists of a set of mandatory and optional attributes. Every entry in the LDAP directory has an object class associated with it.

Before the DB2 database manager can store information in LDAP, the Directory Schema for the LDAP server must include the object classes and attributes that the DB2 database system uses. The process of adding new object classes and attributes to the base schema is called *schema extension*.

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 111. 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.

LDAP support and DB2 Connect

If LDAP support is available at the DB2 Connect gateway, and the database is not found at the gateway database directory, then the DB2 database manager will look up the database location in LDAP and will attempt to keep the found information.

Registering host databases in LDAP

When you register host databases in LDAP, there are two possible configurations: direct connection to the host databases or, connection to the host database though a gateway.

About this task

For direct connection to the host databases, you register the host server in LDAP, then catalog the host database in LDAP specifying the node name of the host server. For connection to the host database though a gateway, you register the gateway server in LDAP, then catalog the host database in LDAP specifying the node name of the gateway server.

If LDAP support is available at the DB2 Connect gateway, and the database is not found at the gateway database directory, the DB2 database system looks up LDAP and attempts to keep the found information.

Procedure

In general, you can manually configure host database information in LDAP so that each client is not required to manually catalog the database and node locally on each computer. The process is as follows:

1. Register the host database server in LDAP.

   You must specify the remote computer name, instance name, and the node type for the host database server in the **REGISTER** command using the **REMOTE**, **INSTANCE**, and **NODETYPE** clauses. The **REMOTE** clause can be set to either the host name or the LU name of the host server machine. The **INSTANCE** clause can be set to any character string that has eight characters or less. (For example, the instance name can be set to DB2.) The **NODETYPE** clause must be set to DCS to indicate that this is a host database server.
2. Register the host database in LDAP using the **CATALOG LDAP DATABASE** command.

   Any additional DRDA parameters can be specified by using the **PARMS** clause. The database authentication type should be set to SERVER.

Example

The following example shows both a direct connection to the host databases and a connection to the host database though a gateway. There is a host database called NIAGARA_FALLS, which can accept incoming connections using TCP/IP. If the client cannot connect directly to the host because it does not have DB2 Connect, then it connects using a gateway called goto@niagara.

The following steps must be done:

1. Register the host database server in LDAP for TCP/IP connectivity. The TCP/IP hostname of the server is "myhost" and the port number is "446". The **NODETYPE** clause is set to DCS to indicate that this is a host database server.

   ```
   db2 register ldap as nftcpip tcpip hostname myhost svcename 446
      remote mvssys instance mvsinst nodetype dcs
   ```

2. Register a DB2 Connect gateway server in LDAP for TCP/IP connectivity. The TCP/IP hostname for the gateway server is "niagara" and the port number is "50000".

   ```
   db2 register ldap as whasf tcpip hostname niagara svcename 50000
      remote niagara instance goto nodetype server
   ```

3. Catalog the host database in LDAP using TCP/IP connectivity. The host database name is "NIAGARA_FALLS", the database alias name is "nftcpip". The **GWNODE** clause is used to specify the nodename of the DB2 Connect gateway server.

   ```
   db2 catalog ldap database NIAGARA_FALLS as nftcpip at node nftcpip
      gwnode whasf authentication server
   ```

After completing the registration and cataloging shown previously, if you want to connect to the host using TCPIP, you connect to nftcpip. If you do not have DB2 Connect on your client workstation, the connection goes through the gateway using TCPIP. From the gateway, it connects to the host using TCP/IP.

Extending the directory schema for IBM Tivoli Directory Server

If you are using IBM Tivoli Directory Server, all the object classes and attributes that are required by the DB2 database before Version 8.2 are included in the base schema.

Run the following command to extend the base schema with new DB2 database attributes introduced in Version 8.2, and later:

```
ldapmodify -c -h machine_name:389 -D dn -w password -f altgwnode.ldif
```

The following is the content of the altgwnode.ldif file:

```
dn:                cn=schema
changetype:        modify
add:               attributetypes
attributetypes:    (
 1.3.18.0.2.4.3092
 NAME 'db2altgwPtr'
 DESC 'DN pointer to DB2 alternate gateway (node) object'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12)
-
add:               ibmattributetypes
ibmattributetypes: (
 1.3.18.0.2.4.3092
 DBNAME ('db2altgwPtr' 'db2altgwPtr')
 ACCESS-CLASS NORMAL
 LENGTH 1000)


dn:                cn=schema
changetype:        modify
add:               attributetypes
attributetypes:    (
 1.3.18.0.2.4.3093
 NAME 'db2altnodePtr'
 DESC 'DN pointer to DB2 alternate node object'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12)
```

```
-
add:               ibmattributetypes
ibmattributetypes: (
 1.3.18.0.2.4.3093
 DBNAME ('db2altnodePtr' 'db2altnodePtr')
 ACCESS-CLASS NORMAL
 LENGTH 1000)


dn:            cn=schema
changetype:    modify
replace:       objectclasses
objectclasses: (
 1.3.18.0.2.6.117
 NAME 'DB2Database'
 DESC 'DB2 database'
 SUP cimSetting
 MUST ( db2databaseName $ db2nodePtr )
 MAY ( db2additionalParameters $ db2altgwPtr $ db2altnodePtr
    $ db2ARLibrary $ db2authenticationLocation $ db2databaseAlias
    $ db2databaseRelease $ db2gwPtr $ DCEPrincipalName ) )
```

The `altgwnode.ldif` and `altgwnode.readme` files can be found at URL: ftp://ftp.software.ibm.com/ps/products/db2/tools/ldap

After adding the DB2 schema definition, the Directory Server must be restarted for all changes to be active.

Netscape LDAP directory support and attribute definitions

The supported level for Netscape LDAP Server is Version 4.12, or later.

Within Netscape LDAP Server Version 4.12, or later, the Netscape Directory Server allows applications to extend the schema by adding attribute and object class definitions to the following two files, `slapd.user_oc.conf` and `slapd.user_at.conf`. These two files are located in the `<Netscape_install path>\slapd-<machine_name>\config` directory.

Note: If you are using Sun One Directory Server 5.0, refer to the topic about extending the directory schema for the Sun One Directory Server.

The DB2 attributes must be added to the `slapd.user_at.conf` as follows:

```
########################################################################
#
# IBM DB2 Database
# Attribute Definitions
#
# bin -> binary
# ces -> case exact string
# cis -> case insensitive string
# dn -> distinguished name
#
########################################################################

attribute binProperty                   1.3.18.0.2.4.305     bin
attribute binPropertyType               1.3.18.0.2.4.306     cis
attribute cesProperty                   1.3.18.0.2.4.307     ces
attribute cesPropertyType               1.3.18.0.2.4.308     cis
attribute cisProperty                   1.3.18.0.2.4.309     cis
attribute cisPropertyType               1.3.18.0.2.4.310     cis
attribute propertyType                  1.3.18.0.2.4.320     cis
attribute systemName                    1.3.18.0.2.4.329     cis
attribute db2nodeName                   1.3.18.0.2.4.419     cis
```

```
 attribute db2nodeAlias                    1.3.18.0.2.4.420     cis
 attribute db2instanceName                 1.3.18.0.2.4.428     cis
 attribute db2Type                         1.3.18.0.2.4.418     cis
 attribute db2databaseName                 1.3.18.0.2.4.421     cis
 attribute db2databaseAlias                1.3.18.0.2.4.422     cis
 attribute db2nodePtr                      1.3.18.0.2.4.423     dn
 attribute db2gwPtr                        1.3.18.0.2.4.424     dn
 attribute db2additionalParameters         1.3.18.0.2.4.426     cis
 attribute db2ARLibrary                    1.3.18.0.2.4.427     cis
 attribute db2authenticationLocation       1.3.18.0.2.4.425     cis
 attribute db2databaseRelease              1.3.18.0.2.4.429     cis
 attribute DCEPrincipalName                1.3.18.0.2.4.443     cis
```

The DB2 object classes must be added to the `slapd.user_oc.conf` file as follows:

```
#########################################################################
#
# IBM DB2 Database
# Object Class Definitions
#
#########################################################################

objectclass eProperty
        oid 1.3.18.0.2.6.90
        requires
                objectClass
        allows
                cn,
                propertyType,
                binProperty,
                binPropertyType,
                cesProperty,
                cesPropertyType,
                cisProperty,
                cisPropertyType

objectclass eApplicationSystem
        oid 1.3.18.0.2.6.84
        requires
                objectClass,
                systemName


objectclass DB2Node
        oid 1.3.18.0.2.6.116
        requires
                objectClass,
                db2nodeName
        allows
                db2nodeAlias,
                host,
                db2instanceName,
                db2Type,
                description,
                protocolInformation

objectclass DB2Database
        oid 1.3.18.0.2.6.117
        requires
                objectClass,
                db2databaseName,
                db2nodePtr
        allows
                db2databaseAlias,
                description,
                db2gwPtr,
                db2additionalParameters,
```

```
db2authenticationLocation,
DCEPrincipalName,
db2databaseRelease,
db2ARLibrary
```

After adding the DB2 schema definition, the Directory Server must be restarted for all changes to be active.

Extending the directory schema for Sun One Directory Server

The Sun One Directory Server is also known as the Netscape or iPlanet directory server.

To have the Sun One Directory Server work in your environment, add the `60ibmdb2.ldif` file to the following directory:

On Windows, if you have iPlanet installed in `C:\iPlanet\Servers`, add the previously mentioned file to `.\sldap-<machine_name>\config\schema`.

On UNIX, if you have iPlanet installed in `/usr/iplanet/servers`, add the previously mentioned file to `./slapd-<machine_name>/config/schema`.

The following is the contents of the file:

```
########################################################################
# IBM DB2 Database
########################################################################
dn: cn=schema
########################################################################
# Attribute Definitions (Before V8.2)
########################################################################
attributetypes: ( 1.3.18.0.2.4.305  NAME 'binProperty'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.5              X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.306  NAME 'binPropertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.307  NAME 'cesProperty'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.308  NAME 'cesPropertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.309  NAME 'cisProperty'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.310  NAME 'cisPropertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.320  NAME 'propertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.329  NAME 'systemName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.419  NAME 'db2nodeName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.420  NAME 'db2nodeAlias'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.428  NAME 'db2instanceName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.418  NAME 'db2Type'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.421  NAME 'db2databaseName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.422  NAME 'db2databaseAlias'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.426  NAME 'db2additionalParameters'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.427  NAME 'db2ARLibrary'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.425  NAME 'db2authenticationLocation'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
```

```
attributetypes: ( 1.3.18.0.2.4.429  NAME 'db2databaseRelease'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.443  NAME 'DCEPrincipalName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.423  NAME 'db2nodePtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.424  NAME 'db2gwPtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
#############################################################################
# Attribute Definitions (V8.2 and later)
#############################################################################
attributetypes: ( 1.3.18.0.2.4.3092 NAME 'db2altgwPtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12              X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.3093 NAME 'db2altnodePtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12              X-ORIGIN 'IBM DB2' )
#############################################################################
# Object Class Definitions
# DB2Database for V8.2 has the above two new optional attributes.
#############################################################################
objectClasses: ( 1.3.18.0.2.6.90  NAME 'eProperty'
   SUP top STRUCTURAL MAY ( cn $ propertyType $ binProperty
   $ binPropertyType $ cesProperty $ cesPropertyType $ cisProperty
   $ cisPropertyType ) X-ORIGIN 'IBM DB2' )
objectClasses: ( 1.3.18.0.2.6.84  NAME 'eApplicationSystem'
   SUP top STRUCTURAL MUST systemName
   X-ORIGIN 'IBM DB2' )
objectClasses: ( 1.3.18.0.2.6.116 NAME 'DB2Node'
   SUP top STRUCTURAL MUST db2nodeName MAY ( db2instanceName $ db2nodeAlias
   $ db2Type $ description $ host $ protocolInformation )
   X-ORIGIN 'IBM DB2' )
objectClasses: ( 1.3.18.0.2.6.117 NAME 'DB2Database'
   SUP top STRUCTURAL MUST (db2databaseName $ db2nodePtr ) MAY
   ( db2additionalParameters $ db2altgwPtr $ db2altnodePtr $ db2ARLibrary
   $ db2authenticationLocation $ db2databaseAlias $ db2databaseRelease
   $ db2gwPtr $ DCEPrincipalName $ description )
   X-ORIGIN 'IBM DB2' )
```

The `60ibmdb2.ldif` and `60ibmdb2.readme`files can be found at URL: ftp://ftp.software.ibm.com/ps/products/db2/tools/ldap

After adding the DB2 schema definition, the Directory Server must be restarted for all changes to be active.

Windows Active Directory

The DB2 database servers are published in the Active Directory as the ibm_db2Node objects. The ibm_db2Node object class is a subclass of the ServiceConnectionPoint (SCP) object class.

Each ibm_db2Node object contains protocol configuration information to allow client applications to connect to the DB2 database server. When a new database is created, the database is published in the Active Directory as the ibm_db2Database object under the ibm_db2Node object.

When connecting to a remote database, a DB2 client queries the Active Directory, through the LDAP interface, for the ibm_db2Database object. The protocol communication to connect to the database server (binding information) is obtained from the ibm_db2Node object, which the ibm_db2Database object is created under.

Property pages for the ibm_db2Node and ibm_db2Database objects can be viewed or modified using the Active Directory Users and Computer Management Console

(MMC) at a domain controller. To set up the property page, run the **regsrv32** command to register the property pages for the DB2 objects as follows:

```
regsvr32 %DB2PATH%\bin\db2ads.dll
```

You can view the objects by using the Active Directory Users and Computer Management Console (MMC) at a domain controller. To get to this administration tool, follow **Start** > **Program** > **Administration Tools** > **Active Directory Users and Computer**.

Note: You must select **Users, Groups, and Computers as containers** from the **View** menu to display the DB2 database objects under the computer objects.

Note: If the DB2 database system is not installed on the domain controller, you can still view the property pages of DB2 database objects by copying the `db2ads.dll` file from `%DB2PATH%\bin` and the resource DLL `db2adsr.dll` from `%DB2PATH%\msg\locale-name` to a local directory on the domain controller. (The directory where you place these two copied files must be one of the directories found in the **PATH** environment variable.) Then, you run the **regsvr32** command from the local directory to register the DLL.

Configuring the DB2 database manager to use Active Directory

In order to access Microsoft Active Directory, some prerequisites need to be met.

Before you begin

- The machine that runs DB2 database must belong to a Windows 2000 or Windows Server 2003 domain.
- The Microsoft LDAP client must be installed. The Microsoft LDAP client is part of the Windows 2000, Windows XP, and Windows Server 2003 operating systems.

Procedure

1. Enable LDAP support.
2. Log on to a domain user account when running the DB2 database system to read information from the Active Directory.

Security considerations for Active Directory

The DB2 database and node objects are created under the computer object of the machine where the DB2 server is installed in the Active Directory. To register a database server or to catalog a database in the Active Directory, you must have sufficient access to create or update the objects under the computer object.

By default, objects under the computer object are readable by any authenticated users and can be updated by administrators (users that belong to the Administrators, Domain Administrators, and Enterprise Administrators groups). To grant access for a specific user or a group, use the Active Directory Users and Computer Management Console (MMC) as follows:

1. Start the Active Directory Users and Computer administration tool:

   Start > **Program** > **Administration Tools** > **Active Directory Users and Computer**.
2. Under **View**, select **Advanced Features**.
3. Select the **Computers** container.
4. Right click the computer object that represents the server machine where DB2 is installed and select **Properties**.

5. Select the **Security** tab, then add the required access to the specified user or group.

The DB2 registry variables and CLI settings at the user level are maintained in the DB2 property object under the user object. To set the DB2 registry variables or CLI settings at the user level, a user needs to have sufficient access to create objects under the User object.

By default, only administrators have access to create objects under the User object. To grant access to a user to set the DB2 registry variables or CLI settings at the user level, use the Active Directory Users and Computer Management Console (MMC) as follows:

1. Start the Active Directory Users and Computer administration tool:

   Start > **Program** > **Administration Tools** > **Active Directory Users and Computer**.
2. Select the user object under the **Users** container.
3. Right click the user object and select **Properties**.
4. Select the **Security** tab.
5. Add the user name to the list by using the **Add** button.
6. Grant "Write", and "Create All Child Objects" access.
7. Using the **Advanced** setting, set permissions to apply to "This object and all child objects".
8. Select the check box **Allow inheritable permissions from parent to propagate to this object**.

DB2 objects in the Active Directory

The DB2 database manager creates objects in the Active Directory at two locations:

1. The DB2 database and node objects are created under the computer object of the machine where the DB2 server is installed. For the DB2 server that does not belong to the Windows domain, the DB2 database and node objects are created under the "System" container.
2. The DB2 registry variables and CLI settings at the user level are stored in the DB2 property objects under the User object. These objects contain information that is specific to that user.

Extending the directory schema for Active Directory

Before the DB2 database manager can store information in the Active Directory, the directory schema needs to be extended to include the new DB2 database object classes and attributes. The process of adding new object classes and attributes to the directory schema is called *schema extension*.

About this task

You must extend the schema for Active Directory by running the DB2 Schema Installation program, **db2schex**. You should run this command before installing DB2 products and creating databases, otherwise you have to manually register the node and catalog the databases.

The **db2schex** program is included on the product CD-ROM in the following location: `x:\db2\windows\utilities\` where x: is the DVD drive letter.

To update the schema, you must be a member of the Schema Administrators group or have been delegated the rights to update the schema. Run the following command on any machine that is part of the Windows domain:

```
runas /user:MyDomain\Administrator x:\db2\Windows\utilities\db2schex.exe
```

where `x:` represents the DVD drive letter.

If you have run the **db2schex** command in an earlier version of the DB2 database management system, when you run this same command again on DB2 UDB Version 8.2, or later, the following two optional attributes are added to the ibm-db2Database class:

```
ibm-db2AltGwPtr
ibm-db2NodePtr
```

If you have not run the **db2schex** command on an earlier version of the DB2 database management system on Windows, when you run this same command on DB2 Version 9.7, or later, all the classes and attributes for DB2 database system LDAP support are added.

There are other optional clauses associated with this command. For more information, refer to the "db2schex - Active Directory schema extension command" topic.

Examples:

- To install the DB2 database schema:

  ```
  db2schex
  ```

- To install the DB2 database schema and specify a bind DN and password:

  ```
  db2schex -b "cn=A Name,dc=toronto1,dc=ibm,dc=com"
           -w password
  ```

  Or,

  ```
  db2schex -b Administrator -w password
  ```

- To uninstall the DB2 database schema:

  ```
  db2schex -u
  ```

- To uninstall the DB2 database schema and ignore errors:

  ```
  db2schex -u -k
  ```

Enabling LDAP support after installation is complete

Before you can use LDAP, you must enable it after the DB2 database product installation is complete.

Procedure

To enable LDAP support:

1. On any machine that is part of a Windows domain, perform the following steps:
   a. If you did not do so before installing the DB2 database product, you must extend the directory schema if you want to use Microsoft Active Directory. For more information, see the "Extending the directory schema for Active Directory" topic.
   b. Install the LDAP support binary files by running the DB2 Setup program and selecting the LDAP Directory Exploitation support from Custom install.

The **Setup** program sets automatically the DB2 registry variable **DB2_ENABLE_LDAP** to YES which is a required setting to enable LDAP support.

c. Optional: To use the IBM LDAP client instead of the Microsoft LDAP client, set the **DB2LDAP_CLIENT_PROVIDER** registry variable to IBM.

2. On each LDAP client, perform the following steps:

   a. Specify the TCP/IP host name and optionally the port number of the LDAP server by running the following command: `db2set DB2LDAPHOST=base_domain_name[:port_number]` where *base_domain_name* is the TCP/IP host name, and *[:port_number]* is the port number. If you do not specify a port number, the default LDAP port number 389 is used. For an SSL enabled LDAP server, run the following command: `db2set DB2LDAPHOST=base_domain_name:SSL:636` where *base_domain_name* is the TCP/IP host name.

      DB2 objects are located in the LDAP base distinguished name (baseDN). You can configure the baseDN on each machine by running the following command:

      ```
      db2set DB2LDAP_BASEDN=baseDN
      ```

      where *baseDN* is the name of the LDAP suffix that is defined at the LDAP server.

   b. Optional: To use LDAP to store DB2 user-specific information, enter the distinguished name (DN) and password of the LDAP user.

3. If you extended the directory schema after installing the DB2 database product, perform the following steps:

   a. Register the current instance of the DB2 server in LDAP by running the following command:

      ```
      db2 register ldap as node-name protocol tcpip
      ```

   b. Register specific databases in LDAP by running the following command:

      ```
      db2 catalog ldap database dbname as alias_dbname
      ```

What to do next

You can now register the LDAP entries.

Note: Retrieving catalog information from an LDAP server using SSL is currently an unsupported feature. To ensure data exchanged between the client and server are encrypted with SSL, see *Configuring Secure Sockets Layer (SSL) support in the DB2 client* for more information.

Registering LDAP entries

Registration of DB2 servers after installation

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.

Catalog a node alias for ATTACH

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
```

Registration of databases in the LDAP directory

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.

Deregistering LDAP entries

Deregistering the DB2 server

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.

Configuring LDAP users

Creating an LDAP user

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 the LDAP user 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.

Disabling LDAP support

Procedure

To disable LDAP support:

1. For each instance of the DB2 server, deregister the DB2 server from LDAP:

   ```
   db2 deregister db2 server in ldap node nodename
   ```

2. Set the DB2 profile registry variable **DB2_ENABLE_LDAP** to NO.

Updating the protocol information for the DB2 server

The DB2 server information in LDAP must be kept current. For example, changes to the protocol configuration parameters or the server network address require an update to LDAP.

About this task

Examples of protocol configuration parameters that can be updated include a TCP/IP host name and service name or port number parameters.

Procedure

- To update the DB2 server in LDAP on the local computer, use the **UPDATE LDAP** command.
- To update remote DB2 server protocol configuration parameters, use the **UPDATE LDAP** command with a **node** clause:

  ```
  db2 update ldap
     node node_name
     hostname host_name
     svcename tcpip_service_name
  ```

Rerouting LDAP clients to another server

Just as with the ability to reroute clients on a system failure, the same ability is also available to you when working with LDAP.

Before you begin

The DB2_ENABLE_LDAP registry variable must be set to “Yes”.

About this task

Within an LDAP environment, all database and node directory information is maintained at an LDAP server. The client retrieves information from the LDAP directory. This information is updated in its local database and node directories if the DB2LDAPCACHE registry variable is set to “Yes”.

Use the **UPDATE ALTERNATE SERVER FOR LDAP DATABASE** command to define the alternate server for a database that represents the DB2 database in LDAP. Alternatively, you can call the db2LdapUpdateAlternateServerForDB API from a client application to update the alternate server for the database in LDAP.

Once established, this alternate server information is returned to the client upon connection.

Note: It is strongly recommended to keep the alternate server information stored in the LDAP server synchronized with the alternate server information stored at the database server instance. Issuing the **UPDATE ALTERNATE SERVER FOR DATABASE** command (notice that it is not "FOR LDAP DATABASE") at the database server instance will help ensure synchronization between the database server instance and the LDAP server.

When you enter **UPDATE ALTERNATE SERVER FOR DATABASE** command at the server instance, and if LDAP support is enabled (DB2_ENABLE_LDAP=Yes) on the server, and if the LDAP user ID and password is cached (**db2ldcfg** was previously run), then the alternate server for the database is automatically, or implicitly, updated on the LDAP server. This works as if you entered UPDATE ALTERNATE SERVER FOR LDAP DATABASE explicitly.

If the **UPDATE ALTERNATE SERVER FOR LDAP DATABASE** command is issued from an instance other than the database server instance, ensure the alternate server information is also identically configured at the database server instance using the **UPDATE ALTERNATE SERVER FOR DATABASE** command. After the client initially connects to the database server instance, the alternate server information returned from the database server instance will take precedence over what is configured in the LDAP server. If the database server instance has no alternate server information configured, client reroute will be considered disabled after the initial connect.

Attaching to a remote server in the LDAP environment

In the LDAP environment, you can attach to a remote database server using the LDAP node name on the **ATTACH** command: `db2 attach to` *`ldap_node_name`*.

About this task

When a client application attaches to a node or connects to a database for the first time, since the node is not in the local node directory, the database manager searches the LDAP directory for the target node entry. If the entry is found in the LDAP directory, the protocol information of the remote server is retrieved. If you connect to the database and if the entry is found in the LDAP directory, then the database information is also retrieved. Using this information, the database manager automatically catalogs a database entry and a node entry on the local computer. The next time the client application attaches to the same node or database, the information in the local database directory is used without having to search the LDAP directory.

A caching mechanism exists so that the client searches the LDAP server only once. After the information is retrieved, it is stored or cached on the local computer based on the values of the **dir_cache** database manager configuration parameter and the **DB2LDAPCACHE** registry variable.

- If **DB2LDAPCACHE**=NO and **dir_cache**=NO, then always read the information from LDAP.

- If **DB2LDAPCACHE**=NO and **dir_cache**=YES, then read the information from LDAP once and insert it into the DB2 cache.
- If **DB2LDAPCACHE**=YES or is not set, then read the information from LDAP server once and cache it into the local database, node, and DCS directories.

Note: The caching of LDAP information is not applicable to user-level CLI or DB2 profile registry variables.

Refreshing LDAP entries in local database and node directories

The DB2 database system provides a caching mechanism to reduce the number of times a client searches the LDAP server.

About this task

After the information is retrieved, it is stored or cached on the local computer based on the values of the **dir_cache** database manager configuration parameter and the **DB2LDAPCACHE** registry variable.

- If **DB2LDAPCACHE**=NO and **dir_cache**=NO, then always read the information from LDAP.
- If **DB2LDAPCACHE**=NO and **dir_cache**=YES, then read the information from LDAP once and insert it into the DB2 cache.
- If **DB2LDAPCACHE**=YES or is not set, then read the information from LDAP server once and cache it into the local database, node, and DCS directories.

Note: The caching of LDAP information is not applicable to user-level CLI or DB2 profile registry variables. Since information in LDAP is subject to change, it might be necessary to refresh the LDAP entries cached in the local database and node directories. There are a few ways to do this.

Procedure

- To refresh all the local database and node entries that were retrieved from LDAP, use the following command:

  ```
  db2 refresh ldap immediate
  ```

- To refresh existing local database and node entries and add new entries from LDAP, use the following command:

  ```
  db2 refresh ldap immediate all
  ```

  Specifying the **IMMEDIATE ALL** parameter adds all the NODE and DB entries contained with the LDAP server into the local directories.

- Alternatively, to force DB2 for Linux, UNIX, and Windows to refresh the database entries that refer to LDAP resources on the next database connection or instance attachment, use the following command:

  ```
  db2 refresh ldap database directory
  ```

- Likewise, to force the DB2 database manager to refresh the nodes entries that refer to LDAP resources on the next database connection or instance attachment, use the following command:

  ```
  db2 refresh ldap node directory
  ```

Results

As part of the refresh, all the LDAP entries that are saved in the local database and node directories are removed. The next time that the application accesses the database or node, it will read the information directly from LDAP and generate a

new entry in the local database or node directory.

What to do next

To ensure that the refresh is done in a timely way, you might want to:

- Schedule a refresh that is run periodically.
- Run the **REFRESH** command during system bootup.
- Use an available administration package to invoke the **REFRESH** command on all client computers.
- Set **DB2LDAPCACHE**=NO to avoid LDAP information being cached in the database, node, and DCS directories.

Searching the LDAP servers

The DB2 database system searches the current LDAP server but in an environment where there are multiple LDAP servers, you can define the scope of the search.

About this task

For example, if the information is not found in the current LDAP server, you can specify automatic search of all other LDAP servers, or, alternatively, you can restrict the search scope to only the current LDAP server, or to the local DB2 database catalog.

When you set the search scope, this sets the default search scope for the entire enterprise. The search scope is controlled through the DB2 database profile registry variable, **DB2LDAP_SEARCH_SCOPE**. To set the search scope value, use the **-gl** option, which means global in LDAP, on the **db2set** command:

```
db2set -gl db2ldap_search_scope=value
```

Possible values include: `local`, `domain`, or `global`. If it is not set, the default value is `domain` which limits the search scope to the directory on the current LDAP server.

For example, you might want to initially set the search scope to `global` after a new database is created. This allows any DB2 client configured to use LDAP to search all the LDAP servers to find the database. Once the entry has been recorded on each computer after the first connect or attach for each client, if you have caching enabled, the search scope can be changed to `local`. Once changed to `local`, each client will not scan any LDAP servers.

Note: The DB2 database profile registry variables **DB2LDAP_KEEP_CONNECTION** and **DB2LDAP_SEARCH_SCOPE** are the only registry variables that can be set at the global level in LDAP.

Chapter 22. SQL and XML limits

The tables in this topic describe SQL and XML limits. Adhering to the most restrictive case can help you to design application programs that are easily portable.

Table 112 lists limits in bytes. These limits are enforced after conversion from the application code page to the database code page when creating identifiers. The limits are also enforced after conversion from the database code page to the application code page when retrieving identifiers from the database. If, during either of these processes, the identifier length limit is exceeded, truncation occurs or an error is returned.

Character limits vary depending on the code page of the database and the code page of the application. For example, because the width of a UTF-8 character can range from 1 to 4 bytes, the character limit for an identifier in a Unicode table whose limit is 128 bytes will range from 32 to 128 characters, depending on which characters are used. If an attempt is made to create an identifier whose name is longer than the limit for this table after conversion to the database code page, an error is returned.

Applications that store identifier names must be able to handle the potentially increased size of identifiers after code page conversion has occurred. When identifiers are retrieved from the catalog, they are converted to the application code page. Conversion from the database code page to the application code page can result in an identifier becoming longer than the byte limit for the table. If a host variable declared by the application cannot store the entire identifier after code page conversion, it is truncated. If that is unacceptable, the host variable can be increased in size to be able to accept the entire identifier name.

The same rules apply to DB2 utilities retrieving data and converting it to a user-specified code page. If a DB2 utility, such as export, is retrieving the data and forcing conversion to a user-specified code page (using the export CODEPAGE modifier or the **DB2CODEPAGE** registry variable), and the identifier expands beyond the limit that is documented in this table because of code page conversion, an error might be returned or the identifier might be truncated.

Table 112. Identifier Length Limits

Description	Maximum in Bytes
Alias name	128
Attribute name	128
Audit policy name	128
Authorization name (can only be single-byte characters)	128
Buffer pool name	18
Column name[2]	128
Constraint name	128
Correlation name	128
Cursor name	128
Data partition name	128

Table 112. Identifier Length Limits (continued)

Description	Maximum in Bytes
Data source column name	255
Data source index name	128
Data source name	128
Data source table name (*remote-table-name*)	128
Database partition group name	128
Database partition name	128
Event monitor name	128
External program name	128
Function mapping name	128
Group name	128
Host identifier[1]	255
Identifier for a data source user (*remote-authorization-name*)	128
Identifier in an SQL procedure (condition name, for loop identifier, label, result set locator, statement name, variable name)	128
Index name	128
Index extension name	18
Index specification name	128
Label name	128
Namespace uniform resource identifier (URI)	1000
Nickname	128
Package name	128
Package version ID	64
Parameter name	128
Password to access a data source	32
Procedure name	128
Role name	128
Savepoint name	128
Schema name[2]	128
Security label component name	128
Security label name	128
Security policy name	128
Sequence name	128
Server (database alias) name	8
Specific name	128
SQL condition name	128
SQL variable name	128
Statement name	128
Storage Group	128
Table name	128

Table 112. Identifier Length Limits (continued)

Description	Maximum in Bytes
Table space name	18
Transform group name	18
Trigger name	128
Trusted context name	128
Type mapping name	18
User-defined function name	128
User-defined method name	128
User-defined type name[2]	128
View name	128
Wrapper name	128
XML element name, attribute name, or prefix name	1000
XML schema location uniform resource identifier (URI)	1000

Note:

1. Individual host language compilers might have a more restrictive limit on variable names.
2. The SQLDA structure is limited to storing 30-byte column names, 18-byte user-defined type names, and 8-byte schema names for user-defined types. Because the SQLDA is used in the DESCRIBE statement, embedded SQL applications that use the DESCRIBE statement to retrieve column or user-defined type name information must conform to these limits.

Table 113. Numeric Limits

Description	Limit
Smallest SMALLINT value	-32 768
Largest SMALLINT value	+32 767
Smallest INTEGER value	-2 147 483 648
Largest INTEGER value	+2 147 483 647
Smallest BIGINT value	-9 223 372 036 854 775 808
Largest BIGINT value	+9 223 372 036 854 775 807
Largest decimal precision	31
Maximum exponent (E_{max}) for REAL values	38
Smallest REAL value	-3.402E+38
Largest REAL value	+3.402E+38
Minimum exponent (E_{min}) for REAL values	-37
Smallest positive REAL value	+1.175E-37
Largest negative REAL value	-1.175E-37
Maximum exponent (E_{max}) for DOUBLE values	308
Smallest DOUBLE value	-1.79769E+308
Largest DOUBLE value	+1.79769E+308

Table 113. Numeric Limits (continued)

Description	Limit
Minimum exponent (E_{min}) for DOUBLE values	-307
Smallest positive DOUBLE value	+2.225E-307
Largest negative DOUBLE value	-2.225E-307
Maximum exponent (E_{max}) for DECFLOAT(16) values	384
Smallest DECFLOAT(16) value[1]	-9.999999999999999E+384
Largest DECFLOAT(16) value	9.999999999999999E+384
Minimum exponent (E_{min}) for DECFLOAT(16) values	-383
Smallest positive DECFLOAT(16) value	1.000000000000000E-383
Largest negative DECFLOAT(16) value	-1.000000000000000E-383
Maximum exponent (E_{max}) for DECFLOAT(34) values	6144
Smallest DECFLOAT(34) value[1]	-9.999999999999999999999999999999999E+6144
Largest DECFLOAT(34) value	9.999999999999999999999999999999999E+6144
Minimum exponent (E_{min}) for DECFLOAT(34) values	-6143
Smallest positive DECFLOAT(34) value	1.000000000000000000000000000000000E-6143
Largest negative DECFLOAT(34) value	-1.000000000000000000000000000000000E-6143

Note:

1. These are the limits of normal decimal floating-point numbers. Valid decimal floating-point values include the special values NAN, -NAN, SNAN, -SNAN, INFINITY and -INFINITY. In addition, valid values include subnormal numbers.

   Subnormal numbers are nonzero numbers whose adjusted exponents are less than E_{min}. For a subnormal number, the minimum value of the exponent is E_{min} - (*precision*-1), called E_{tiny}, where *precision* is the working precision (16 or 34). That is, subnormal numbers extend the range of numbers close to zero by 15 or 33 orders of magnitude for DECFLOAT(16) or DECFLOAT(34), respectively. Subnormal numbers are different from normal numbers because the maximum number of digits for a subnormal number is less than the working precision (16 or 34). Decimal floating-point cannot represent the subnormal numbers with the same accuracy as it can represent normal numbers. The smallest positive subnormal number for DECFLOAT(34) is $1x10^{-6176}$, which contains only one digit, whereas the smallest positive normal number for DECFLOAT(34) is $1.000000000000000000000000000000000x10^{-6143}$, which contains 34 digits. The smallest positive subnormal number for DECFLOAT(16) is $1x10^{-398}$.

Table 114. String Limits

Description	Limit
Maximum length of CHAR (in bytes)	254

Table 114. String Limits (continued)

Description	Limit
Maximum length of VARCHAR (in bytes)	32 672
Maximum length of LONG VARCHAR (in bytes)[1]	32 700
Maximum length of CLOB (in bytes)	2 147 483 647
Maximum length of serialized XML (in bytes)	2 147 483 647
Maximum length of GRAPHIC (in double-byte characters)	127
Maximum length of VARGRAPHIC (in double-byte characters)	16 336
Maximum length of LONG VARGRAPHIC (in double-byte characters)[1]	16 350
Maximum length of DBCLOB (in double-byte characters)	1 073 741 823
Maximum length of BLOB (in bytes)	2 147 483 647
Maximum length of character constant	32 672
Maximum length of graphic constant	16 336
Maximum length of concatenated character string	2 147 483 647
Maximum length of concatenated graphic string	1 073 741 823
Maximum length of concatenated binary string	2 147 483 647
Maximum number of hexadecimal constant digits	32 672
Largest instance of a structured type column object at run time (in gigabytes)	1
Maximum size of a catalog comment (in bytes)	254

Note:

1. The LONG VARCHAR and LONG VARGRAPHIC data types are deprecated and might be removed in a future release.

Table 115. XML Limits

Description	Limit
Maximum depth of an XML document (in levels)	125
Maximum size of an XML schema document (in bytes)	31 457 280

Table 116. Datetime Limits

Description	Limit
Smallest DATE value	0001-01-01
Largest DATE value	9999-12-31
Smallest TIME value	00:00:00
Largest TIME value	24:00:00
Smallest TIMESTAMP value	0001-01-01-00.00.00.000000000000
Largest TIMESTAMP value	9999-12-31-24.00.00.000000000000
Smallest timestamp precision	0
Largest timestamp precision	12

Table 117. Database Manager Limits

Category	Description	Limit
Applications	Maximum number of host variable declarations in a precompiled program[3]	storage
	Maximum length of a host variable value (in bytes)	2 147 483 647
	Maximum number of declared cursors in a program	storage
	Maximum number of rows changed in a unit of work	storage
	Maximum number of cursors opened at one time	storage
	Maximum number of connections per process within a DB2 client	512
	Maximum number of simultaneously opened LOB locators in a transaction	4 194 304
	Maximum size of an SQLDA (in bytes)	storage
	Maximum number of prepared statements	storage
Buffer Pools	Maximum NPAGES in a buffer pool for 32-bit releases	1 048 576
	Maximum NPAGES in a buffer pool for 64-bit releases	2 147 483 647
	Maximum total size of all buffer pool slots (4K)	2 147 483 646
Concurrency	Maximum number of concurrent users of a server[4]	64 000
	Maximum number of concurrent users per instance	64 000
	Maximum number of concurrent applications per database	60 000
	Maximum number of databases per instance concurrently in use	256
Constraints	Maximum number of constraints on a table	storage
	Maximum number of columns in a UNIQUE constraint (supported through a UNIQUE index)	64
	Maximum combined length of columns in a UNIQUE constraint (supported through a UNIQUE index, in bytes)[8]	8192
	Maximum number of referencing columns in a foreign key	64
	Maximum combined length of referencing columns in a foreign key (in bytes)[8]	8192
	Maximum length of a check constraint specification (in bytes)	65 535
Databases	Maximum database partition number	999
	Maximum members in a DB2 pureScale environment	128

Table 117. Database Manager Limits (continued)

Category	Description	Limit
Indexes	Maximum number of indexes on a table	32 767 or storage
	Maximum number of columns in an index key	64
	Maximum length of an index key including all overhead[6] [8]	*indexpagesize*/4
	Maximum length of a variable index key part (in bytes)[7]	1022 or storage
	Maximum size of an index per database partition in an SMS table space (in terabytes)[6]	64
	Maximum size of an index per database partition in a regular DMS table space (in gigabytes)[6]	512
	Maximum size of an index per database partition in a large DMS table space (in terabytes)[6]	64
	Maximum length of a variable index key part for an index over XML data (in bytes)[7]	*pagesize*/4 - 207
Log records	Maximum Log Sequence Number	0xFFFF FFFF FFFF FFFF
Monitoring	Maximum number of simultaneously active event monitors	128
	In a DB2 partitioned database environment, maximum number of simultaneously active GLOBAL event monitors	32
Routines	Maximum number of parameters in a procedure with LANGUAGE SQL	32 767
	Maximum number of parameters in an external procedure with PROGRAM TYPE MAIN	32 767
	Maximum number of parameters in an external procedure with PROGRAM TYPE SUB	90
	Maximum number of parameters in a cursor value constructor	32 767
	Maximum number of parameters in a user-defined function	90
	Maximum number of nested levels for routines	64
	Maximum number of schemas in the SQL path	64
	Maximum length of the SQL path (in bytes)	2048

Table 117. Database Manager Limits (continued)

Category	Description	Limit
Security	Maximum number of elements in a security label component of type set or tree	64
	Maximum number of elements in a security label component of type array	65 535
	Maximum number of security label components in a security policy	16
SQL	Maximum total length of an SQL statement (in bytes)	2 097 152
	Maximum number of tables referenced in an SQL statement or a view	storage
	Maximum number of host variable references in an SQL statement	32 767
	Maximum number of constants in a statement	storage
	Maximum number of elements in a select list[6]	1012
	Maximum number of predicates in a WHERE or HAVING clause	storage
	Maximum number of columns in a GROUP BY clause[6]	1012
	Maximum total length of columns in a GROUP BY clause (in bytes)[6]	32 677
	Maximum number of columns in an ORDER BY clause[6]	1012
	Maximum total length of columns in an ORDER BY clause (in bytes)[6]	32 677
	Maximum level of subquery nesting	storage
	Maximum number of subqueries in a single statement	storage
	Maximum number of values in an insert operation[6]	1012
Storage Groups	Maximum number of storage groups in a database	256
	Maximum number of storage paths in a storage group	128
	Maximum length of a storage path (in bytes)	175

Table 117. Database Manager Limits (continued)

Category	Description	Limit
Tables and Views	Maximum number of columns in a table [6]	1012
	Maximum number of columns in a view[1]	5000
	Maximum number of columns in a data source table or view that is referenced by a nickname	5000
	Maximum number of columns in a distribution key[5]	500
	Maximum length of a row including all overhead[2] [6]	32 677
	Maximum number of rows in a non-partitioned table, per database partition	128×10^{10}
	Maximum number of rows in a data partition, per database partition	128×10^{10}
	Maximum size of a table per database partition in a regular table space (in gigabytes)[3] [6]	512
	Maximum size of a table per database partition in a large DMS table space (in terabytes)[6]	64
	Maximum number of data partitions for a single table	32 767
	Maximum number of table partitioning columns	16
	Maximum number of fields in a user-defined row type	1012
Table Spaces	Maximum size of a LOB object per table or per table partition (in terabytes)	4
	Maximum size of a LF object per table or per table partition (in terabytes)	2
	Maximum number of table spaces in a database	32 768
	Maximum number of tables in an SMS table space	65 532
	Maximum size of a regular DMS table space (in gigabytes) [3] [6]	512
	Maximum size of a large DMS table space (in terabytes) [3] [6]	64
	Maximum size of a temporary DMS table space (in terabytes) [3] [6]	64
	Maximum number of table objects in a DMS table space	See Table 118 on page 538
Triggers	Maximum runtime depth of cascading triggers	16
User-defined Types	Maximum number of attributes in a structured type	4082

Table 117. Database Manager Limits (continued)

Category	Description	Limit
Workload Manager	Maximum number of user-defined service superclasses per database	64
	Maximum number of user-defined service subclasses per service superclass	61

Note:

1. This maximum can be achieved using a join in the CREATE VIEW statement. Selecting from such a view is subject to the limit of most elements in a select list.
2. The actual data for BLOB, CLOB, LONG VARCHAR, DBCLOB, and LONG VARGRAPHIC columns is not included in this count. However, information about the location of that data does take up some space in the row.
3. The numbers shown are architectural limits and approximations. The practical limits may be less.
4. The actual value is controlled by the **max_connections** and **max_coordagents** database manager configuration parameters.
5. This is an architectural limit. The limit on the most columns in an index key should be used as the practical limit.
6. For page size-specific values, see Table 118.
7. This is limited only by the longest index key, including all overhead (in bytes). As the number of index key parts increases, the maximum length of each key part decreases.
8. The maximum can be less, depending on index options.

Table 118. Database Manager Page Size-specific Limits

Description	4K page size limit	8K page size limit	16K page size limit	32K page size limit
Maximum number of table objects in a DMS table space[1]	51 971[2] 53 212[3]	53 299	53 747	54 264
Maximum number of columns in a table	500	1012	1012	1012
Maximum length of a row including all overhead	4005	8101	16 293	32 677
Maximum size of a table per database partition in a regular table space (in gigabytes)	64	128	256	512
Maximum size of a table per database partition in a large DMS table space (in terabytes)	8	16	32	64
Maximum length of an index key including all overhead (in bytes)	1024	2048	4096	8192
Maximum size of an index per database partition in an SMS table space (in terabytes)	8	16	32	64
Maximum size of an index per database partition in a regular DMS table space (in gigabytes)	64	128	256	512

Table 118. Database Manager Page Size-specific Limits (continued)

Description	4K page size limit	8K page size limit	16K page size limit	32K page size limit
Maximum size of an index per database partition in a large DMS table space (in terabytes)	8	16	32	64
Maximum size of a regular DMS table space (in gigabytes)	64	128	256	512
Maximum size of a large DMS table space (in terabytes)	8	16	32	64
Maximum size of a temporary DMS table space (in terabytes)	8	16	32	64
Maximum number of elements in a select list	500[4]	1012	1012	1012
Maximum number of columns in a GROUP BY clause	500	1012	1012	1012
Maximum total length of columns in a GROUP BY clause (in bytes)	4005	8101	16 293	32 677
Maximum number of columns in an ORDER BY clause	500	1012	1012	1012
Maximum total length of columns in an ORDER BY clause (in bytes)	4005	8101	16 293	32 677
Maximum number of values in an insert operation	500	1012	1012	1012
Maximum number of SET clauses in a single update operation	500	1012	1012	1012
Maximum records per page for a regular table space	251	253	254	253
Maximum records per page for a large table space	287	580	1165	2335

Note:

1. Table objects include table data, indexes, LONG VARCHAR columns, LONG VARGRAPHIC columns, and LOB columns. Table objects that are in the same table space as the table data do not count extra toward the limit. However, each table object that is in a different table space than the table data does contribute one toward the limit for each table object type per table in the table space in which the table object resides.
2. When extent size is 2 pages.
3. When extent size is any size other than 2 pages.
4. In cases where the only system temporary table space is 4KB and the data overflows to the sort buffer, an error is generated. If the result set can fit into memory, there is no error.

Chapter 23. Registry and environment variables

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 119. 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).
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 120. 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`

Table 120. Common commands for setting registry variables (continued)

Desired Action	Command
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 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.

Option	Description

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`

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`

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*

Aggregate registry variables

Use an aggregate registry variable to group several registry variables as a configuration that is identified by another registry variable name. Each registry variable that is part of the group has a predefined setting. The aggregate registry variable is given a value that is interpreted as declaring several registry variables.

The intention of an aggregate registry variable is to ease registry configuration for broad operational objectives.

The only valid aggregate registry variable is **DB2_WORKLOAD**.

Valid values for this variable are:

- 1C
- CM
- COGNOS_CS
- FILENET_CM
- INFOR_ERP_LN
- MAXIMO
- MDM
- SAP
- TPM
- WAS
- WC
- WP

Any registry variable that is implicitly configured through an aggregate registry variable might also be explicitly defined. Explicitly setting a registry variable that was previously given a value through the use of an aggregate registry variable is useful when doing performance or diagnostic testing. Explicitly setting a variable that is configured implicitly by an aggregate is referred to as overriding the variable.

If you attempt to modify an explicitly set registry variable by using an aggregate registry variable, a warning is issued and the explicitly set value is kept. This warning tells you that the explicit value is maintained. If the aggregate registry variable is used first and then you specify an explicit registry variable, a warning is not given.

When you query the aggregate registry variable, only the value assigned to that variable is shown. Most users should not care about the values for each individual variable.

The following example shows the interaction between using the aggregate registry variable and explicitly setting a registry variable. To control your database environment, you might set the **DB2_WORKLOAD** aggregate registry variable to SAP and override the **DB2_SKIPDELETED** registry variable to NO. By running the **db2set** command, you receive the following results:

```
DB2_WORKLOAD=SAP
DB2_SKIPDELETED=NO
```

In another situation, you might set **DB2ENVLIST**, set the **DB2_WORKLOAD** aggregate registry variable to SAP, and override the **DB2_SKIPDELETED** registry variable to NO. When you issue the **db2set** command, the registry variables that were configured automatically by setting the aggregate registry variable show the name of the aggregate displayed in square brackets, next to its value. The **DB2_SKIPDELETED** registry variable shows a NO value, with [O] displayed next to its value.

When you no longer require the configuration that is associated with **DB2_WORKLOAD**, delete the implicit values of each registry variable in the group by deleting the value of the aggregate registry variable. Use the following command to delete the value of the **DB2_WORKLOAD** variable:

```
db2set DB2_WORKLOAD=
```

After deleting the **DB2_WORKLOAD** aggregate registry variable value, restart the database. After the database is restarted, the registry variables that were implicitly configured by the aggregate registry variable are no longer in effect.

Deleting the value of an aggregate registry variable does not delete the value for a registry variable that was set explicitly. It does not matter that the registry variable is a member of the group definition that was deleted. The explicit setting for the registry variable is maintained.

You might need to see the values for each registry variable that is a member of the **DB2_WORKLOAD** aggregate registry variable. For instance, you might want to see the values that would be used if you configured **DB2_WORKLOAD** to SAP. To find the values that would be used if **DB2_WORKLOAD**=SAP, run `db2set -gd DB2_WORKLOAD=SAP`.

DB2 registry and environment variables

DB2 database products provide a number of registry variables and environment variables that you might need to know about to get up and running.

To view a list of all supported registry variables, execute the following command:

```
db2set -lr
```

You must set values for registry variables that you want to update before you execute the **db2start** command.

The following table lists all registry variables by category.

Table 121. Registry and environment variables summary. The second column is split into two columns.

Variable category	Registry or environment variable name	
General	DB2ACCOUNT DB2BIDI DB2_CAPTURE_LOCKTIMEOUT DB2CODEPAGE DB2_COLLECT_TS_REC_INFO DB2_CONNRETRIES_INTERVAL DB2CONSOLECP DB2DBDFT DB2DISCOVERYTIME DB2_ENFORCE_MEMBER_SYNTAX DB2FODC DB2_FORCE_APP_ON_MAX_LOG	DB2GRAPHICUNICODESERVER DB2INCLUDE DB2INSTDEF DB2INSTOWNER DB2_LIC_STAT_SIZE DB2LOCALE DB2_MAX_CLIENT_CONNRETRIES DB2_OBJECT_TABLE_ENTRIES DB2_SYSTEM_MONITOR_SETTINGS DB2TERRITORY DB2_VIEW_REOPT_VALUES
System environment	DB2_ALTERNATE_GROUP_LOOKUP DB2CONNECT_ENABLE_EURO_CODEPAGE DB2CONNECT_IN_APP_PROCESS DB2_COPY_NAME DB2_CPU_BINDING DB2DBMSADDR DB2_DIAGPATH DB2DOMAINLIST DB2ENVLIST DB2INSTANCE DB2INSTPROF DB2LDAPSecurityConfig DB2LIBPATH DB2LOGINRESTRICTIONS	DB2NODE DB2OPTIONS DB2_PARALLEL_IO DB2PATH DB2_PMAP_COMPATIBILITY DB2PROCESSORS DB2RCMD_LEGACY_MODE DB2RESILIENCE DB2_RESTORE_GRANT_ADMIN_AUTHORITIES DB2SYSTEM DB2_UPDDBCFG_SINGLE_DBPARTITION DB2_USE_PAGE_CONTAINER_TAG DB2_WORKLOAD
Communications	DB2CHECKCLIENTINTERVAL DB2COMM DB2FCMCOMM DB2_FORCE_NLS_CACHE DB2_PMODEL_SETTINGS DB2RSHCMD DB2RSHTIMEOUT	DB2SORCVBUF DB2SOSNDBUF DB2TCP_CLIENT_CONTIMEOUT DB2TCP_CLIENT_KEEPALIVE_TIMEOUT DB2TCP_CLIENT_RCVTIMEOUT DB2TCPCONNMGRS DB2TCP_SERVER_KEEPALIVE_TIMEOUT
Command-line	DB2BQTIME DB2BQTRY DB2_CLP_EDITOR DB2_CLP_HISTSIZE	DB2_CLPPROMPT DB2IQTIME DB2RQTIME
Partitioned database environment	DB2CHGPWD_EEE DB2_FCM_SETTINGS DB2_FORCE_OFFLINE_ADD_PARTITION	DB2_NUM_FAILOVER_NODES DB2_PARTITIONEDLOAD_DEFAULT DB2PORTRANGE
DB2 pureScale environment	DB2_DATABASE_CF_MEMORY	DB2_MCR_RECOVERY_PARALLELISM_CAP
Query compiler	DB2_ANTIJOIN DB2_DEFERRED_PREPARE_SEMANTICS DB2_INLIST_TO_NLJN DB2_LIKE_VARCHAR DB2_MINIMIZE_LISTPREFETCH	DB2_NEW_CORR_SQ_FF DB2_OPT_MAX_TEMP_SIZE DB2_REDUCED_OPTIMIZATION DB2_SELECTIVITY DB2_SQLROUTINE_PREPOPTS

Table 121. Registry and environment variables summary (continued). The second column is split into two columns.

Variable category	Registry or environment variable name	
Performance	DB2_ALLOCATION_SIZE	DB2_MMAP_WRITE
	DB2_APM_PERFORMANCE	DB2_NO_FORK_CHECK
	DB2ASSUMEUPDATE	DB2NTMEMSIZE
	DB2_AVOID_PREFETCH	DB2NTNOCACHE
	DB2_BACKUP_USE_DIO	DB2NTPRICLASS
	DB2BPVARS	DB2NTWORKSET
	DB2CHKPTR	DB2_OVERRIDE_BPF
	DB2CHKSQLDA	DB2_PINNED_BP
	DB2_EVALUNCOMMITTED	DB2PRIORITIES
	DB2_EXTENDED_IO_FEATURES	DB2_RCT_FEATURES
	DB2_EXTENDED_OPTIMIZATION	DB2_RESOURCE_POLICY
	DB2_IO_PRIORITY_SETTING	DB2_SELUDI_COMM_BUFFER
	DB2_KEEP_AS_AND_DMS_CONTAINERS_OPEN	DB2_SET_MAX_CONTAINER_SIZE
	DB2_KEEPTABLELOCK	DB2_SKIPDELETED
	DB2_LARGE_PAGE_MEM	DB2_SKIPINSERTED
	DB2_LOGGER_NON_BUFFERED_IO	DB2_SMS_TRUNC_TMPTABLE_THRESH
	DB2MAXFSCRSEARCH	DB2_SORT_AFTER_TQ
	DB2_MAX_INACT_STMTS	DB2_SQLWORKSPACE_CACHE
	DB2_MAX_NON_TABLE_LOCKS	DB2_TRUSTED_BINDIN
	DB2_MDC_ROLLOUT	DB2_USE_ALTERNATE_PAGE_CLEANING
	DB2MEMDISCLAIM	DB2_USE_FAST_PREALLOCATION
	DB2_MEM_TUNING_RANGE	DB2_USE_IOCP
	DB2_MMAP_READ	

Table 121. Registry and environment variables summary (continued). The second column is split into two columns.

Variable category	Registry or environment variable name	
Miscellaneous	**DB2ADMINSERVER**	**DB2_INDEX_PCTFREE_DEFAULT**
	DB2_ATS_ENABLE	**DB2LDAP_BASEDN**
	DB2AUTH	**DB2LDAPCACHE**
	DB2_BCKP_INCLUDE_LOGS_WARNING	**DB2LDAP_CLIENT_PROVIDER**
	DB2_BCKP_PAGE_VALIDATION	**DB2LDAPHOST**
	DB2CLIINIPATH	**DB2LDAP_KEEP_CONNECTION**
	DB2_COMMIT_ON_EXIT	**DB2LDAP_SEARCH_SCOPE**
	DB2_COMMON_APP_DATA_PATH	**DB2_LOAD_COPY_NO_OVERRIDE**
	DB2_COMPATIBILITY_VECTOR	**DB2_LIMIT_FENCED_GROUP**
	DB2_CREATE_DB_ON_PATH	**DB2LOADREC**
	DB2_DDL_SOFT_INVAL	**DB2LOCK_TO_RB**
	DB2_DISABLE_FLUSH_LOG	**DB2_MAX_LOB_BLOCK_SIZE**
	DB2_DISPATCHER_PEEKTIMEOUT	**DB2_MEMORY_PROTECT**
	DB2_DJ_INI	**DB2_MIN_IDLE_RESOURCES**
	DB2_DMU_DEFAULT	**DB2_NCHAR_SUPPORT**
	DB2_DOCHOST	**DB2NOEXITLIST**
	DB2_DOCPORT	**DB2_NUM_CKPW_DAEMONS**
	DB2DSDRIVER_CFG_PATH	**DB2_OPTSTATS_LOG**
	DB2DSDRIVER_CLIENT_HOSTNAME	**DB2REMOTEPREG**
	DB2_ENABLE_AUTOCONFIG_DEFAULT	**DB2_RESOLVE_CALL_CONFLICT**
	DB2_ENABLE_LDAP	**DB2_RESTRICT_DDF**
	DB2_EVMON_EVENT_LIST_SIZE	**DB2_SAS_SETTINGS**
	DB2_EVMON_STMT_FILTER	**DB2SATELLITEID**
	DB2_EXTSECURITY	**DB2_SERVER_CONTIMEOUT**
	DB2_FALLBACK	**DB2_SERVER_ENCALG**
	DB2_FMP_COMM_HEAPSZ	**DB2SORT**
	DB2_GRP_LOOKUP	**DB2_STANDBY_ISO**
	DB2_HADR_BUF_SIZE	**DB2STMM**
	DB2_HADR_NO_IP_CHECK	**DB2_TRUNCATE_REUSESTORAGE**
	DB2_HADR_PEER_WAIT_LIMIT	**DB2_UTIL_MSGPATH**
	DB2_HADR_ROS	**DB2_XBSA_LIBRARY**
	DB2_HADR_SORCVBUF	**DB2_XSLT_ALLOWED_PATH**
	DB2_HADR_SOSNDBUF	
	DB2_HISTORY_FILTER	

General registry variables

You will find it useful to know about the general registry variables in the DB2 environment, especially if you plan on altering them in the future. Please note that some registry variables listed here apply to specific operating system environments.

DB2ACCOUNT

- Operating system: All
- Default: NULL
- This variable defines the accounting string that is sent to the remote host. Refer to the DB2 Connect User's Guide for details.

DB2BIDI

- Operating system: All
- Default: NO, Values: YES or NO
- This variable enables bidirectional support and the **DB2CODEPAGE** variable is used to declare the code page to be used.

DB2_CAPTURE_LOCKTIMEOUT

- Operating system: All
- Default: NULL, Values: ON or NULL
- This variable specifies to log descriptive information about lock timeouts at the time that they occur. The logged information identifies: the key applications involved in the lock contention that resulted in the lock timeout, the details about what these applications were running at the time of the lock timeout, and the details about the lock causing the contention. Information is captured for both the lock requestor (the application that received the lock timeout error) and the current lock owner. A text report is written and stored in a file for each lock timeout.

  The files are created using the following naming convention: db2locktimeout.*par*.*AGENTID*.*yyyy-mm-dd-hh-mm-ss*, where *par* is the database partition number; *AGENTID* is the Agent ID; *yyyy-mm-dd-hh-mm-ss* is the timestamp consisting of the year, month, day, hour, minute and second. In non-partitioned database environments, *par* is set to 0.

  The location of the file is based on the value set in the **diagpath** database configuration parameter. If **diagpath** is not set, then the file is located in one of the following directories:

  – In Windows environments:
    - If you do not set the **DB2INSTPROF** environment variable, information is written to *x*:*SQLLIB**DB2INSTANCE*, where *x* is the drive reference, *SQLLIB* is the directory that you specified for the **DB2PATH** registry variable, and *DB2INSTANCE* is the name of the instance owner.
    - If you set the **DB2INSTPROF** environment variable, information is written to *x*:*DB2INSTPROF**DB2INSTANCE*, where *x* is the drive reference, *DB2INSTPROF* is the name of the instance profile directory, and *DB2INSTANCE* is the name of the instance owner.
    - If you set the **DB2INSTPROF** environment variable to a new location, you must ensure that it contains the appropriate files and folders to run the instance. This may require you to copy all of the files and folders from the previous location to the new location.
  – In Linux and UNIX environments: information is written to *INSTHOME*/sqllib/db2dump, where *INSTHOME* is the home directory of the instance.

  Delete lock timeout report files when you no longer need them. Because the report files are in the same location as other diagnostics logs, the DB2 system could shutdown if the directory is allowed to get full. If you need to keep some lock timeout report files, move them to a directory or folder different than where the DB2 logs are stored.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

Important: This variable is deprecated and might be removed in a future release because there are new methods to collect lock timeout events using the CREATE EVENT MONITOR FOR LOCKING statement.

DB2CODEPAGE

- Operating system: All

- Default: derived from the language ID, as specified by the operating system.
- This variable specifies the code page of the data presented to DB2 for database client application. The user should not set **DB2CODEPAGE** unless explicitly stated in DB2 documents, or asked to do so by DB2 service. Setting **DB2CODEPAGE** to a value not supported by the operating system can produce unexpected results. Normally, you do not need to set **DB2CODEPAGE** because DB2 automatically derives the code page information from the operating system.

  Note: Because Windows does not report a Unicode code page (in the Windows regional settings) instead of the ANSII code page, a Windows application will not behave as a Unicode client. To override this behavior, set the **DB2CODEPAGE** registry variable to 1208 (for the Unicode code page) to cause the application to behave as a Unicode application.

DB2_COLLECT_TS_REC_INFO

- Operating system: All
- Default: ON, Values: ON or OFF
- This variable specifies whether DB2 will process all log files when rolling forward a table space, regardless of whether the log files contain log records that affect the table space. To skip the log files known not to contain any log records affecting the table space, set this variable to ON. **DB2_COLLECT_TS_REC_INFO** must be set before the log files are created and used so that the information required for skipping log files is collected.

DB2_CONNRETRIES_INTERVAL

- Operating system: All
- Default: Not set, Values: an integer number of seconds
- This variable specifies the sleep time between consecutive connection retries, in seconds, for the automatic client reroute feature. You can use this variable in conjunction with **DB2_MAX_CLIENT CONNRETRIES** to configure the retry behavior for automatic client reroute.

  If **DB2_MAX_CLIENT_CONNRETRIES** is set, but **DB2_CONNRETRIES_INTERVAL** is not, **DB2_CONNRETRIES_INTERVAL** defaults to 30. If **DB2_MAX_CLIENT_CONNRETRIES** is not set, but **DB2_CONNRETRIES_INTERVAL** is set, **DB2_MAX_CLIENT_CONNRETRIES** defaults to 10. If neither **DB2_MAX_CLIENT_CONNRETRIES** nor **DB2_CONNRETRIES_INTERVAL** is set, the automatic client reroute feature reverts to its default behavior of retrying the connection to a database repeatedly for up to 10 minutes.

DB2CONSOLECP

- Operating system: Windows
- Default: NULL, Values: all valid code page values
- Specifies the code page for displaying DB2 message text. When specified, this value overrides the operating system code page setting.

DB2DBDFT

- Operating system: All
- Default: NULL
- This variable specifies the database alias name of the database to be used for implicit connects. If an application has no database connection

but SQL or XQuery statements are issued, an implicit connect will be made if the **DB2DBDFT** environment variable has been defined with a default database.

DB2DISCOVERYTIME

- Operating system: Windows
- Default: 40 seconds, Minimum: 20 seconds
- This variable specifies the amount of time that SEARCH discovery will search for DB2 systems.

DB2_ENFORCE_MEMBER_SYNTAX

- Operating system: All
- Default: OFF , Values: OFF or ON
- This variable allows you to control whether or not the syntax for SQL statements, DB2 commands, and APIs will be checked for the correct usage of the database partition keywords to determine whether the MEMBER keyword must be used instead. In a DB2 pureScale environment, the default behavior is to tolerate the usage of keywords specific to database partitions, such as DBPARTITIONNUM or DATABASE PARTITION, even when the operation is targeting a DB2 member. However, when **DB2_ENFORCE_MEMBER_SYNTAX** is set to ON, the MEMBER keyword must be specified correctly, otherwise SQL1538N is returned. The setting of this variable is ignored and has no effect outside of a DB2 pureScale environment.

DB2_EXPRESSION_RULES

- Operating system: All
- Default: Empty, Values: RAISE_ERROR_PERMIT_SKIP or RAISE_ERROR_PERMIT_DROP
- The settings for the **DB2_EXPRESSION_RULES** registry variable control how the DB2 Optimizer determines the access plan for queries which involve a RAISE_ERROR function. The default behaviour of the RAISE_ERROR function is that no filtering may be pushed beyond the expression containing this function. This can result in no predicates being applied during the table accesses which can lead to excessive computation of expressions, excessive locking and poor query performance.

  In certain cases this behaviour is too strict, depending on the particular business requirements of the application, it may not matter if predicates and joins are applied before the application of RAISE_ERROR. For example in the context of a row level security implementation, there is typically an expression of the form:

  ```
  CASE WHEN <conditions for validatin access to this row>
       THEN NULL
       ELSE RAISE_ERROR(...)
  END
  ```

  The application may only be concerned with validating access to the rows which are selected by the query and not in validating access to every row in the table. Thus predicates could be applied in the base table access and the expression containing the RAISE_ERROR only needs to executed after all the filtering is performed. In this case a value of **DB2_EXPRESSION_RULES**=RAISE_ERROR_PERMIT_SKIP may be appropriate.

  Another alternative is in the context of COLUMN LEVEL security. In this case there are typically expressions of the form:

```
CASE WHEN <conditions for validating access to this row and column>
     THEN <table.column>
     ELSE RAISE_ERROR(...)
END
```

In this case the application may only want errors to be raised if the user attempts to receive the data for a particular row and column contains a value that the user is not allowed to retrieve. In this case a setting of **DB2_EXPRESSION_RULES**=RAISE_ERROR_PERMIT_DROP will only cause the expression containing the RAISE_ERROR function to be evaluated if the particular column is used by a predicate or a column function, or if it is returned as output from the query.

DB2FODC

- Operating system: All
- Default: The concatenation of all FODC parameters (see following list)
  - for Linux and UNIX: "CORELIMIT=*val* DUMPCORE=ON DUMPDIR=*diagpath*"
  - for Windows: "DUMPDIR=*diagpath*"

  Note that the parameters are separated by spaces.
- This registry variable controls a set of troubleshooting-related parameters used in First Occurrence Data Collection (FODC). Use **DB2FODC** to control different aspects of data collection in outage situations.

  This registry variable is read once, during the DB2 instance startup. To perform updates to the FODC parameters online, use **db2pdcfg** tool. Use the **DB2FODC** registry variable to sustain the configuration across reboots. You do not need to specify all of the parameters, nor do you need to specify them in a particular order. The default value is assigned to any parameter that is not specified. For example, if you don't want the core files dumped, but you do want the other parameters' default behaviors, you would issue the command:

  ```
  db2set DB2FODC="DUMPCORE=OFF"
  ```

  Parameters:

  CORELIMIT
  - Operating system: Linux and UNIX
  - Default: Current ulimit setting, Values: `0` to `unlimited`
  - This option specifies the maximum size, in bytes, of core files created. This value overrides the current core file size limit setting. Consideration should be given to the available file system space because core files can be quite large. The size is dependent on the DB2 configuration and the state of the process at the time the problem occurs.

    If **CORELIMIT** is set, DB2 will use this value to override the current user core limit (ulimit) setting to generate the core file.

    If **CORELIMIT** is not set, DB2 will set the core file size to the value equal to the current ulimit setting.

    Note: Any changes to the user core limit or **CORELIMIT** are not effective until the next recycling of the DB2 instance.

  COS
  - Operating system: All

- Default: ON, Values: ON or OFF
- This option specifies if the **db2cos** script is enabled or not. You can use the following parameters with this parameter:

  COS_SLEEP
  - Default: 3, Values: 0 to unlimited
  - This option specifies the amount of time to sleep in seconds between checking the size of the output file generated.

  COS_TIMEOUT
  - Default: 30, Values: 0 to unlimited
  - This option specifies the amount of time to wait in seconds before the script is finished.

  COS_COUNT
  - Default: 255, Values: 0 to 255
  - This option specifies the number of times to execute **db2cos** during a database manager trap.

  COS_SQLO_SIG_DUMP
  - Default: ON, Values: ON or OFF
  - This option specifies if **db2cos** is enabled when the **SQLO_SIG_DUMP** signal is received.

DUMPCORE
- Operating system: Linux, Solaris, AIX
- Default: AUTO, Values: AUTO, ON, or OFF
- This option specifies if core file generation is to take place. Core files, which are used for problem determination and are created in the **diagpath** directory, contain the entire process image of the terminating DB2 process. However, whether or not an actual core file dump occurs depends on the current ulimit setting and value of the **CORELIMIT** parameter. Some operating systems also have configuration settings for core dumps, which may dictate the behavior of application core dumping. The AUTO setting causes a core file to be generated if a trap cannot be sustained when the **DB2RESILIENCE** registry variable is set to ON. The **DUMPCORE**=ON setting always generates a core file by overriding the **DB2RESILIENCE** registry variable setting.

  The recommended method for disabling core file dumps is to set **DUMPCORE** to OFF.

DUMPDIR
- Operating system: All
- Default: **diagpath** directory, or the default diagnostic directory if **diagpath** is not defined, Values: *path to directory*
- This option specifies the absolute path name of the directory for core file creation.

FODCPATH
- Operating system: All
- Default: path defined by the **DIAGPATH** database manager configuration parameter, Values: *fodc_path_name*

– This option specifies the absolute path name of where the FODC package is to be directed. The *fodc_path_name* must be an existing directory and must be writable by the member or members for which it is set for and by the fmp processes running on those members.

SERVICELEVEL

– Operating system: All

– Default: AUTOMATIC ulimit setting, Values: AUTOMATIC, BASIC, or FULL

– This option specifies how data is collected during panics, traps, or errors that might indicate data corruption. DB2 is designed to generate diagnostics that are appropriate to the configuration and problem context. For example, when a trap can be sustained, only the minimum essential diagnostics are generated in order to rollback the transaction and respond to the application as soon as possible, releasing resources which other applications may be waiting on. When a trap cannot be sustained, diagnostics such as db2cos data collection scripts and core dumps may be limited in favor of availability in DB2 pureScale configurations. The default behaviour for generating diagnostics is represented by the SERVICELEVEL setting of AUTOMATIC.

  The following option are supported for this parameter:

  AUTOMATIC
  This setting specifies that the effective SERVICELEVEL setting (that is, BASIC or FULL) is to be chosen at runtime, for the members, and at start time, for the CF process. At present, the only times that BASIC is chosen are for DB2 pureScale environments that have multiple members and for trap resilience.

  BASIC This SERVICELEVEL setting specifies that a minimal amount of FODC data is to be dumped. Core dump processing is disabled by default (but can be overridden by the COREDUMP setting), diagnostics are restricted to the affected thread only, and callout scripts are disabled.

  FULL This SERVICELEVEL setting specifies that the maximum amount of FODC data is to be dumped. This includes core dumps, any associated components dumps, and the invocation of the callout scripts. In addition, there is no attempt to sustain traps.

DB2_FORCE_APP_ON_MAX_LOG

- Operating system: All
- Default: TRUE, Values: TRUE or FALSE
- Specifies what happens when the **max_log** configuration parameter value is exceeded. If set to TRUE, the application is forced off the database and the unit of work is rolled back.

  If FALSE, the current statement fails. The application can still commit the work completed by previous statements in the unit of work, or it can roll back the work completed to undo the unit of work.

Note: This DB2 registry variable affects the ability of the import utility to recover from log full situations. If **DB2_FORCE_APP_ON_MAX_LOG** is set to TRUE and you issue an **IMPORT** command with the **COMMITCOUNT** command option, the import utility will not be able to perform a commit in order to avoid running out of active log space. When the import utility encounters an SQL0964C (Transaction Log Full), it will be forced off the database and the current unit of work will be rolled back.

- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2GRAPHICUNICODESERVER

- Operating system: All
- Default: OFF, Values: ON or OFF
- This registry variable is used to accommodate existing applications written to insert graphic data into a Unicode database. Its use is only needed for applications that specifically send sqldbchar (graphic) data in Unicode instead of the code page of the client. (sqldbchar is a supported SQL data type in C and C++ that can hold a single double-byte character.) When set to ON, you are telling the database that graphic data is coming in Unicode, and the application expects to receive graphic data in Unicode.

DB2INCLUDE

- Operating system: All
- Default: Current directory
- Specifies a path to be used during the processing of the SQL INCLUDE text-file statement during **DB PREP** processing. It provides a list of directories where the INCLUDE file might be found. Refer to Developing Embedded SQL Applications for descriptions of how **DB2INCLUDE** is used in the different precompiled languages.

DB2INSTDEF

- Operating system: Windows
- Default: DB2
- This variable sets the value to be used if **DB2INSTANCE** is not defined.

DB2INSTOWNER

- Operating system: Windows
- Default: **NULL**
- The registry variable created in the DB2 profile registry when the instance is first created. This variable is set to the name of the instance-owning machine.

DB2_LIC_STAT_SIZE

- Operating system: All
- Default: NULL, Range: 0 to 32767
- This variable determines the maximum size (in MBs) of the file containing the license statistics for the system. A value of zero turns the license statistic gathering off. If the variable is not recognized or not defined, the variable defaults to unlimited. The statistics are displayed using the License Center.

DB2LOCALE

- Operating system: All

- Default: NO, Values: YES or NO
- This variable specifies whether the default "C" locale of a process is restored to the default "C" locale after calling DB2 and whether to restore the process locale back to the original 'C' after calling a DB2 function. If the original locale was not 'C', then this registry variable is ignored.

DB2_MAX_CLIENT_CONNRETRIES

- Operating system: All
- Default: Not set, Values: an integer number of maximum times to retry the connection
- This variable specifies the maximum number of connection retries that the automatic client reroute feature will attempt. You can use this variable in conjunction with **DB2_CONNRETRIES_INTERVAL** to configure the retry behavior for automatic client reroute.

  If **DB2_MAX_CLIENT_CONNRETRIES** is set, but **DB2_CONNRETRIES_INTERVAL** is not, **DB2_CONNRETRIES_INTERVAL** defaults to 30. If **DB2_MAX_CLIENT_CONNRETRIES**is not set, but **DB2_CONNRETRIES_INTERVAL** is set, **DB2_MAX_CLIENT_CONNRETRIES** defaults to 10. If neither **DB2_MAX_CLIENT_CONNRETRIES** nor **DB2_CONNRETRIES_INTERVAL** is set, the automatic client reroute feature reverts to its default behavior of retrying the connection to a database repeatedly for up to 10 minutes.

DB2_MAX_GLOBAL_SNAPSHOT_SIZE

- Operating system: All
- Default: Not set, Values: 0 to the maximum size of a snapshot.
- This variable specifies the number of bytes a snapshot or snapshot estimate can be. You can use this variable to prevent large global snapshots from causing memory usage spikes which can cause performance degradation and system hangs.

  By default, **DB2_MAX_GLOBAL_SNAPSHOT_SIZE** is not set, which means an effective limit of the maximum size of a snapshot (2 GB less 512 bytes). If you need to set this variable, a recommended starting point is 25% of the memory allocated for fast communications manager (FCM) buffers. This variable is dynamic and only applies to partitioned database environments.

DB2_OBJECT_TABLE_ENTRIES

- Operating system: All
- Default: 0, Values: 0-65532

  The actual maximum value possible on your system depends on the page size and extent size, but it cannot exceed 65532.
- This variable specifies the expected number of objects in a table space. If you know that a large number of objects (for example, 1000 or more) will be created in a DMS table space, you should set this registry variable to the approximate number before creating the table space. This will reserve contiguous storage for object metadata during table space creation. Reserving contiguous storage reduces the chance that an online backup will block operations which update entries in the metadata (for example, CREATE INDEX, **IMPORT REPLACE**). It will also make resizing the table space easier because the metadata will be stored at the start of the table space.

If the initial size of the table space is not large enough to reserve the contiguous storage, the table space creation will continue without the additional space reserved.

DB2_SYSTEM_MONITOR_SETTINGS

- Operating system: All
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.
- The registry variable controls a set of parameters which allow you to modify the behavior of various aspects of DB2 monitoring. Separate each parameter by a semicolon, as in the following example:

  ```
  db2set DB2_SYSTEM_MONITOR_SETTINGS=OLD_CPU_USAGE:TRUE;
   DISABLE_CPU_USAGE:TRUE
  ```

  Every time you set **DB2_SYSTEM_MONITOR_SETTINGS**, each parameter must be set explicitly. Any parameter that you do not specify when setting this variable reverts back to its default value. So in the following example:

  ```
  db2set DB2_SYSTEM_MONITOR_SETTINGS=DISABLE_CPU_USAGE:TRUE
  ```

  OLD_CPU_USAGE will be restored to its default setting.

  Note: Currently, this registry variable only has settings for Linux; additional settings for other operating systems will be added in future releases.
- Parameters:

  OLD_CPU_USAGE

  - Operating system: Linux
  - Values: TRUE/ON, FALSE/OFF
  - Default value on RHEL4 and SLES9: TRUE (Note: a setting of FALSE for OLD_CPU_USAGE will be ignored-only the old behavior will be used.)
  - Default value on RHEL5, SLES10, and others: FALSE
  - This parameter controls how the instance obtains CPU usage times on Linux platforms. If set to TRUE, the older method of getting CPU usage time is used. This method returns both system and user CPU usage times, but consumes more CPU in doing so (that is, it has a higher overhead). If set to FALSE, the newer method of getting CPU usage is used. This method returns only the user CPU usage value, but is faster because it has less overhead.

  DISABLE_CPU_USAGE

  - Operating system: Linux
  - Values: TRUE/ON, FALSE/OFF
  - Default value on RHEL4 and SLES9: TRUE
  - Default value on RHEL5, SLES10, and others: FALSE
  - This parameter allows you to determine whether CPU usage is read or not. When DISABLE_CPU_USAGE is enabled (set to TRUE), CPU usage is not read, allowing you to avoid the overhead that can sometimes occur during the retrieval of CPU usage.

DB2TERRITORY

- Operating system: All
- Default: derived from the language ID, as specified by the operating system.
- This variable specifies the region, or territory code of the client application, which influences date and time formats.

DB2_VIEW_REOPT_VALUES

- Operating system: All
- Default: NO, Values: YES, NO
- This variable enables all users to store the cached values of a reoptimized SQL or XQuery statement in the EXPLAIN_PREDICATE table when the statement is explained. When this variable is set to NO, only DBADM is allowed to save these values in the EXPLAIN_PREDICATE table.

System environment variables

You use the system environment variables to pass configuration values to running applications in the DB2 environment. Please note that some registry variables listed apply to specific operating system environments.

DB2_ALTERNATE_GROUP_LOOKUP

- Operating system: AIX, Linux
- Default: NULL, Values: NULL, GETGRSET on AIX, GETGROUPLIST on Linux
- This variable allows DB2 database systems to obtain group information from an alternative source provided by the operating system. On AIX, the function getgrset is used. This provides the ability to obtain groups from somewhere other than local files via Loadable Authentication Modules.

DB2_CLP_EDITOR

See DB2_CLP_EDITOR in "Command-line variables" for details.

DB2_CLP_HISTSIZE

See DB2_CLP_HISTSIZE in "Command-line variables" for details.

DB2CONNECT_ENABLE_EURO_CODEPAGE

- Operating system: All
- Default:NO, Values: YES or NO
- Set this variable to YES on all DB2 Connect clients and servers that connect to a DB2 for z/OS server or a DB2 for IBM i server where euro support is required. If you set this variable to YES, the current application code page is mapped to the equivalent coded character set ID (CCSID) that explicitly indicates support for the euro sign. As a result, DB2 Connect connects to the DB2 for z/OS server or DB2 for IBM i server by using a CCSID that is a superset of the CCSID of the current application code and that also supports the euro sign. For example, if the client is using code page that maps to CCSID 1252, the client connects by using CCSID 5348.

DB2CONNECT_IN_APP_PROCESS

- Operating system: All
- Default: YES, Values: YES or NO
- When you set this variable to NO, local DB2 Connect clients on a DB2 Enterprise Server Edition machine are forced to run within an agent.

Some advantages of running within an agent are that local clients can be monitored and that they can use SYSPLEX support.

DB2_COPY_NAME

- Operating system: Windows
- Default: The name of the default copy of DB2 installed on your machine. Values: the name of a copy of DB2 installed on your machine. The name can be up to 128 characters long.
- The **DB2_COPY_NAME** variable stores the name of the copy of DB2 currently in use. If you have multiple DB2 copies installed on your machine, you cannot use **DB2_COPY_NAME** to switch to a different copy of DB2, you must run the command *INSTALLPATH*\bin\db2envar.bat to change the copy currently in use, where *INSTALLPATH* is the location where the DB2 copy is installed.

DB2_CPU_BINDING

- Operating system: Linux
- Default:
  - If a DB2 member and cluster caching facility (CF) are on the same host:
    - For the member, NUM_CORES=max(1, floor(0.8*totalCores))
    - For the cluster caching facility, NUM_CORES=totalCores - the number listed previously.
  - If a DB2 member and cluster caching facility are not sharing a host, this variable is not set
- This registry variable controls CPU pinning. For changes to this variable to take effect, you need to restart the DB2 instance.

  Parameters:

  NUM_CORES
  - Operating system: Linux
  - Default: If the member or CF are on the same host, approximately 80% of the total available cores are assigned to DB2 and the remainder is assigned to the CF. Values: 0 < x < (number of physical cores on the host)
  - This option specifies the number of cores to which the member's or CF's processes are pinned. You can use NUM_CORES to configure sub-capacity licensing of the DB2 product. The number of cores can be a whole number or a fraction, which allows you to add one or more hardware threads if simultaneous multithreading (SMT) is enabled.

  PROCESSOR_LIST
  - Operating system: Linux
  - Default: Not set, Values: any of the processor numbers
  - This option specifies which logical processors DB2 will be bound to, giving you complete control over the number of logical processors (or cores) and which CPU package (or socket) they will reside on. If you try to set both PROCESSOR_LIST and NUM_CORES with **DB2_CPU_BINDING**, NUM_CORES is ignored

  Restart light considerations

  If a member is restarted as a guest member on a host which

already has a member running on it, the restart light member will be pinned to the cores already being used by the resident member, up to the number of cores specified by **DB2_CPU_BINDING**. If a member is restarted as a guest member on a host with fewer cores than what is specified by **DB2_CPU_BINDING**, the member will be bound to the number of cores on the host.

Every time you set **DB2_CPU_BINDING**, any parameter that is not explicitly set is cleared in the instance-level profile. Enclose each parameter and its value in quotes, as shown in the following examples.

Example 1

A user wants to pin the first member (which has a ID of 0) of DB2 instance db2inst1 to one core on a host machine with two cores:

```
db2set -i db2inst1 0 DB2_CPU_BINDING="NUM_CORES=1"
```

Example 2

A user wants to bind all the members in db2inst1 to five logical processors on a host machine with eight cores and Intel HTT enabled (meaning it has 16 logical processors):

```
db2set -i db2inst1 DB2_CPU_BINDING="NUM_CORES=2.5"
```

Example 3

A user wants to specify how many cores the primary CF (which has an ID of 128) is bound to:

```
db2set -i db2inst1 128 DB2_CPU_BINDING="NUM_CORES=4"
```

Example 4

A user wants to bind DB2 for db2inst1 on member 0 to a specific group of logical processors:

```
db2set -i db2inst1 0 DB2_CPU_BINDING="PROCESSOR_LIST=2,10,6,14"
```

DB2DBMSADDR

- Operating system: Linux on x86, Linux on zSeries (31-bit), and Windows 32-bit
- Default: `NULL` on Linux operating systems, `0x20000000` on Windows operating systems, Values: virtual addresses in the range `0x09000000` to `0xB0000000` in increments of 0x10000 on Linux operating systems, `0x20000000` to `0xB0000000` in increments of 0x10000 on Windows operating systems
- The **DB2DBMSADDR** registry variable specifies the default database shared memory address in hexadecimal format.

  This variable can be used to fine tune the address space layout of DB2 processes. This variable changes the location of the instance shared memory from its current location at virtual address 0x10000000 to the new value.

  Note: An incorrect address can cause severe issues with the DB2 database system, ranging from an inability to start a DB2 instance, to an inability to connect to the database. An incorrect address is one that collides with an area in memory that is already in use, or is predestined to be used for something else. To address this problem, reset the **DB2DBMSADDR** registry variable to `NULL` by using the following command:

  ```
  db2set DB2DBMSADDR=
  ```

Note: Before changing the setting of this variable, you must stop the instance and all DB2 processes. If the instance is running while this variable is set, then any subsequent **db2stop** command will fail.

DB2_DIAGPATH

- Operating system: All
- Default: The default value is the instance db2dump directory on UNIX and Linux operating systems, and the instance db2 directory on Windows operating systems.
- This parameter applies to ODBC and CLI applications only.

  This parameter allows you to specify the fully qualified path for DB2 diagnostic information. This directory could possibly contain dump files, trap files, an error log, a notification file, and an alert log file, depending on your platform.

  Setting this environment variable has the same effect for ODBC and CLI applications in the scope of that environment as setting the DB2 database manager configuration parameter **diagpath**, and as setting the CLI/ODBC configuration keyword **DiagPath**.

DB2DOMAINLIST

- Operating system: All
- Default: NULL, Values: A list of Windows domain names separated by commas (",").
- This variable defines one or more Windows domains. The list, which is maintained on the server, defines the domains that the requesting user ID is authenticated against. Only users belonging to these domains have their connection or attachment requests accepted.

  This variable is effective only when CLIENT authentication is set in the database manager configuration. It is needed if a single sign-on from a Windows desktop is required in a Windows domain environment.

  DB2DOMAINLIST is supported if either the client or the server is running in a Windows environment.

DB2ENVLIST

- Operating system: UNIX
- Default: NULL
- This variable lists specific variable names for either stored procedures or user-defined functions. By default, the **db2start** command filters out all user environment variables except those prefixed with "DB2" or "db2". If specific environment variables must be passed to either stored procedures or user-defined functions, you can list the variable names in the **DB2ENVLIST** environment variable. Separate each variable name by one or more spaces.

DB2INSTANCE

- Operating system: All
- Default: **DB2INSTDEF** on Windows 32-bit operating systems.
- This environment variable specifies the instance that is active by default. On UNIX, users must specify a value for **DB2INSTANCE**.

  Note: You cannot use the **db2set** command to update this registry variable. For more information, see "Identifying the current instance" on page 546 and "Setting environment variables outside the profile registries on Windows" on page 544.

DB2INSTPROF

- Operating system: Windows
- Default: `Documents and Settings\All Users\Application Data\IBM\DB2\`*`Copy Name`* (Windows XP, Windows 2003), `ProgramData\IBM\DB2\`*`Copy Name`* (Windows Vista)
- This environment variable specifies the location of the instance directory on Windows operating systems. The instance directory (and other user data files) cannot be under the `sqllib` directory.

DB2LDAPSecurityConfig

- Operating system: All
- Default: `NULL`, Values: valid name and path to the IBM LDAP security plug-in configuration file
- This variable is used to specify the location of the IBM LDAP security plug-in configuration file. If the variable is not set, the IBM LDAP security plug-in configuration file is named `IBMLDAPSecurity.ini` and is in one of the following locations:
  - On Linux and UNIX operating systems: *`INSTHOME`*`/sqllib/cfg/`
  - On Windows operating systems: `%`**`DB2PATH`**`%\cfg\`

  On Windows operating systems, this variable should be set in the global system environment to ensure it is picked up by the DB2 service.

DB2LIBPATH

- Operating system: UNIX
- Default: `NULL`
- DB2 constructs its own shared library path. If you want to add a PATH into the engine's library path (for example, on AIX, a user-defined function requires a specific entry in **LIBPATH**), you must set **DB2LIBPATH**. The actual value of **DB2LIBPATH** is appended to the end of the DB2 constructed shared library path.

DB2LOGINRESTRICTIONS

- Operating system: AIX
- Default: `LOCAL`, Values: `LOCAL`, `REMOTE`, `SU`, `NONE`
- This registry variable allows you to use an AIX operating system API called loginrestrictions(). This API determines whether a user is allowed to access the system. By calling this API, DB2 database security is able to enforce the login restrictions that are specified by the operating system. There are different values that can be submitted to this API when using this registry variable. The values are:
  - `REMOTE`

    DB2 only enforces login restrictions to verify that the account can be used for remote logins through the **rlogind** or **telnetd** programs.
  - `SU`

    DB2 Version 9.1 only enforces **su** restrictions to verify that the **su** command is permitted, and that the current process has a group ID that can invoke the **su** command to switch to the account.
  - `NONE`

    DB2 does not enforce any login restrictions.
  - `LOCAL` (or the variable is not set)

DB2 only enforces login restrictions to verify that local logins are permitted for this account. This is the normal behavior when logging in.

No matter which one of these options you set, user accounts or IDs that have the specified privileges are able to use DB2 successfully both locally on the server and from remote clients. For a description of the loginrestrictions() API, refer to AIX documentation.

DB2NODE

- Operating system: All
- Default: `NULL`, Values: `1` to `999`
- Used to specify the target logical node of a database partition server that you want to attach to or connect to. If this variable is not set, the target logical node defaults to the logical node which is defined with port `0` on the machine. In a partitioned database environment, the connection settings could have an impact on acquiring trusted connections. For example, if the **DB2NODE** variable is set to a node such that the establishment of a connection on that node requires going through an intermediate node (a hop node), it is the IP address of that intermediate node and the communication protocol used to communicate between the hop node and the connection node that are considered when evaluating this connection in order to determine whether or not it can be marked as a trusted connection. In other words, it is not the original node from which the connection was initiated that is considered. Rather, it is the hop node that is considered.

  Note: You cannot use the **db2set** command to update this registry variable. For more information, see "Setting environment variables outside the profile registries on Windows" on page 544.

DB2OPTIONS

- Operating system: All
- Default: `NULL`
- Used to set the command line processor options.

DB2_PARALLEL_IO

- Operating system: All
- Default: `NULL` or `*` (in a DB2 pureScale environment) Values: *TablespaceID*:[*n*],... - a comma-separated list of defined table spaces (identified by their numeric table space ID). If the prefetch size of a table space is AUTOMATIC, you can indicate to the DB2 database manager the number of disks per container for that table space by specifying the table space ID, followed by a colon, followed by the number of disks per container, *n*. If *n* is not specified, the default is `6`.

  You can replace *TablespaceID* with an asterisk (*) to specify all table spaces. For example, if **DB2_PARALLEL_IO**=`*`, all table spaces use six as the number of disks per container. If you specify both an asterisk (*) and a table space ID, the table space ID setting takes precedence. For example, if **DB2_PARALLEL_IO** =`*,1:3`, all table spaces use six as the number of disks per container, except for table space 1, which uses three.
- This registry variable is used to change the way DB2 calculates the I/O parallelism of a table space. When I/O parallelism is enabled (either implicitly, by the use of multiple containers, or explicitly, by setting **DB2_PARALLEL_IO**), it is achieved by issuing the correct number of prefetch requests. Each prefetch request is a request for an extent of

pages. For example, a table space has two containers and the prefetch size is four times the extent size. If the registry variable is set, a prefetch request for this table space will be broken into four requests (one extent per request) with a possibility of four prefetchers servicing the requests in parallel.

You might want to set the registry variable if the individual containers in the table space are striped across multiple physical disks or if the container in a table space is created on a single RAID device that is composed of more than one physical disk.

If this registry variable is not set, the degree of parallelism of any table space is the number of containers of the table space. For example, if **DB2_PARALLEL_IO** is set to NULL and a table space has four containers, four extent-sized prefetch requests are issued; or if a table space has two containers and the prefetch size is four times the extent size, the prefetch request for this table space will be broken into two requests (each request will be for two extents).

If this registry variable is set, and the prefetch size of the table is not AUTOMATIC, the degree of parallelism of the table space is the prefetch size divided by the extent size. For example, if **DB2_PARALLEL_IO** is set for a table space that has a prefetch size of 160 and an extent size of 32 pages, five extent-sized prefetch requests are issued.

If this registry variable is set, and the prefetch size of the table space is AUTOMATIC, DB2 automatically calculates the prefetch size of a table space. The following table summarizes the different options available and how parallelism is calculated for each situation:

Table 122. How Parallelism is Calculated

Prefetch size of table space	DB2_PARALLEL_IO Setting	Parallelism is equal to:
AUTOMATIC	Not set	Number of containers
AUTOMATIC	*Table space ID*	Number of containers * 6
AUTOMATIC	*Table space ID*:*n*	Number of containers * *n*
Not AUTOMATIC	Not set	Number of containers
Not AUTOMATIC	*Table space ID*	Prefetch size/extent size
Not AUTOMATIC	*Table space ID*:*n*	Prefetch size/extent size

Disk contention might result using this variable in some scenarios. For example, if a table space has two containers and each of the two containers have each a single disk dedicated to it, setting the registry variable might result in contention on those disks because the two prefetchers will be accessing each of the two disks at once. However, if each of the two containers was striped across multiple disks, setting the registry variable would potentially allow access to four different disks at once.

To activate changes to this registry variable, issue a **db2stop** command and then enter a **db2start** command.

DB2PATH

- Operating system: Windows

- Default: Varies by operating system
- This environment variable is used to specify the directory where the product is installed on Windows 32-bit operating systems.

DB2_PMAP_COMPATIBILITY

- Operating system: All
- Default: ON, Values: ON or OFF
- This variable allows users to continue using the sqlugtpi and sqlugrpn APIs to return, respectively, the distribution information for a table and the database partition number and database partition server number for a row. The default setting, ON, indicates that the distribution map size remains 4 096 entries (the pre-Version 9.7 behavior). When this variable is set to OFF, the distribution map size for new or upgraded databases is increased to 32 768 entries (the Version 9.7 behavior). If you use the 32K distribution map, you need to use the new db2GetDistMap and db2GetRowPartNum APIs.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2PROCESSORS

- Operating system: Windows
- Default: NULL, Values: 0-*n*-1 (where *n*= the number of processors)
- This variable sets the process affinity mask for a particular **db2syscs** process. In environments running multiple logical nodes, this variable is used to associate a logical node to a processor or set of processors.

  When specified, DB2 issues the SetProcessAffinityMask() api. If unspecified, the **db2syscs** process is associated with all processors on the server.

DB2RCMD_LEGACY_MODE

- Operating system: Windows,
- Default: NULL, Values: YES, ON, TRUE, or 1, or NO, OFF, FALSE, or 0
- This variable allows users to enable or disable the DB2 Remote Command Service's enhanced security. To run the DB2 Remote Command Service in a secure manner, set **DB2RCMD_LEGACY_MODE** to NO, OFF, FALSE, 0, or NULL. To run in legacy mode (without enhanced security), set **DB2RCMD_LEGACY_MODE** to YES, ON, TRUE, or 1. The secure mode is only available if your domain controller is running Windows 2000 or later.

  Note: If **DB2RCMD_LEGACY_MODE** is set to YES, ON, TRUE, or 1, all requests sent to the DB2 Remote Command Service are processed under the context of the requestor. To facilitate this, you must allow either or both the machine and service logon account to impersonate the client by enabling the machine and service logon accounts at the domain controller.

  Note: If **DB2RCMD_LEGACY_MODE** is set to NO, OFF, FALSE, or 0, you must have SYSADM authority in order to have the DB2 Remote Command Service execute commands on your behalf.

DB2RESILIENCE

- Operating system: All

- Default: ON, Values: ON (TRUE or 1), or OFF (FALSE or 0)
- This registry variable can be used to control whether physical read errors are tolerated, and activates extended trap recovery. The default behavior is to tolerate read errors and activate extended trap recovery. To revert to the behavior of previous releases and force the database manager to shutdown the instance, set the registry variable to OFF. This registry variable does not affect the existing storage key support.

DB2_RESTORE_GRANT_ADMIN_AUTHORITIES

- Operating system: All
- Default: OFF, Values: ON or OFF
- If **DB2_RESTORE_GRANT_ADMIN_AUTHORITIES** is set to ON, and you are restoring to a new or existing database, then you will be granted SECADM, DBADM, DATAACCESS, and ACCESSCTRL authorities.
- The following methods of restore are supported when **DB2_RESTORE_GRANT_ADMIN_AUTHORITIES** is set to ON:
  - Split mirror backups
  - ACS Snapshot backups
  - Online and offline database backups with the **RESTORE DATABASE** command

  Note: Note that this variable has no effect on table space restores; no additional authorities will be granted to the user issuing the restore operation.
- If **DB2_WORKLOAD** is set to SAP, **DB2_RESTORE_GRANT_ADMIN_AUTHORITIES** will be set to ON.

DB2SYSTEM

- Operating system: Windows and UNIX
- Default: NULL
- Specifies the name that is used by your users and database administrators to identify the DB2 database server system. If possible, this name should be unique within your network.

  This name aids users in identifying the system that contains the database they wish to access. A value for **DB2SYSTEM** is set at installation time as follows:
  - On Windows the setup program sets it equal to the computer name specified for the Windows system.
  - On UNIX systems, it is set equal to the UNIX system's TCP/IP hostname.

DB2_UPDDBCFG_SINGLE_DBPARTITION

- Operating system: All
- Default: Not set, Values: 0/FALSE/NO, 1/TRUE/YES
- **DB2_UPDDBCFG_SINGLE_DBPARTITION** enables you to revert to the behavior of previous versions of DB2, in which updates to a database configuration apply only to the local database partition or the database partition that is set by the **DB2NODE** registry variable. This allows for backward compatibility support for any existing command scripts or applications that require this behavior.

  When set to 1, TRUE, or, YES, this registry variable allows you to specify that any updates and resets to your database affect only a specific

partition. If the variable is not set (the default), updates or changes to a database configuration act across all database partitions, when you do not specify a partition clause.

Note: This variable does not apply to update or reset requests made by calling ADMIN_CMD routines.

DB2_USE_PAGE_CONTAINER_TAG

- Operating system: All
- Default:NULL, Values: ON, NULL
- By default, DB2 stores a container tag in the first extent of each DMS container, whether it is a file or a device. The container tag is the metadata for the container. Before DB2 Version 8.1, the container tag was stored in a single page, and it thus required less space in the container. To continue to store the container tag in a single page, set **DB2_USE_PAGE_CONTAINER_TAG** to ON.

  However, if you set this registry variable to ON when you use RAID devices for containers, I/O performance might degrade. Because for RAID devices you create table spaces with an extent size equal to or a multiple of the RAID stripe size, setting the **DB2_USE_PAGE_CONTAINER_TAG** to ON causes the extents not to line up with the RAID stripes. As a result, an I/O request might need to access more physical disks than would be optimal. Users are strongly advised against enabling this registry variable unless you have very tight space constraints, or you require behavior consistent with pre-Version 8 databases.

  To activate changes to this registry variable, issue a **db2stop** command and then enter a **db2start** command.

DB2_WORKLOAD

- Operating system: All
- Default: Not set, Values: 1C, CM, COGNOS_CS, FILENET_CM, INFOR_ERP_LN, MAXIMO, MDM, SAP, TPM, WAS, WC, or WP
- Each value for **DB2_WORKLOAD** represents a specific grouping of several registry variables with predefined settings.
- These are the valid values:

  1C Use this setting when you want to configure a set of registry variables in your database for 1C applications.

  CM Use this setting when you want to configure a set of registry variables in your database for IBM Content Manager.

  COGNOS_CS
  Use this setting when you want to configure a set of registry variables in your database for Cognos® Content Server.

  FILENET_CM
  Use this setting when you want to configure a set of registry variables in your database for Filenet Content Manager.

  INFOR_ERP_LN
  Use this setting when you want to configure a set of registry variables in your database for Infor ERP Baan.

  MAXIMO
  Use this setting when you want to configure a set of registry variables in your database for Maximo®.

MDM Use this setting when you want to configure a set of registry variables in your database for Master Data Management.

SAP Use this setting when want to configure a set of registry variables in your database for the SAP environment.

When you have set **DB2_WORKLOAD**=SAP, the user table space SYSTOOLSPACE and the user temporary table space SYSTOOLSTMPSPACE are not automatically created. These table spaces are used for tables created automatically by the following wizards, utilities, or functions:

- Automatic maintenance
- SYSINSTALLOBJECTS stored procedure, if the table space input parameter is not specified
- GET_DBSIZE_INFO stored procedure

Without the SYSTOOLSPACE and SYSTOOLSTMPSPACE table spaces, you cannot use these wizards, utilities, or functions.

To be able to use these wizards, utilities, or functions, do either of the following:

- Manually create the SYSTOOLSPACE table space to hold the objects that the tools need (in a partitioned database environment, create this table space on the catalog partition). For example:

```
CREATE REGULAR TABLESPACE SYSTOOLSPACE
IN IBMCATGROUP
MANAGED BY SYSTEM
USING ('SYSTOOLSPACE')
```

- Specifying a valid table space, call the SYSINSTALLOBJECTS stored procedure to create the objects for the tools, and specify the identifier for the particular tool. SYSINSTALLOBJECTS will create a table space for you. If you do not want to use SYSTOOLSSPACE for the objects, specify a different user-defined table space.

After completing at least one of these choices, create the SYSTOOLSTMPSPACE temporary table space (also on the catalog partition, if you're working in a partitioned database environment). For example:

```
CREATE USER TEMPORARY TABLESPACE SYSTOOLSTMPSPACE
IN IBMCATGROUP
MANAGED BY SYSTEM
USING ('SYSTOOLSTMPSPACE')
```

Once the table space SYSTOOLSPACE and the temporary table space SYSTOOLSTMPSPACE are created, you can use the wizards, utilities, or functions mentioned earlier.

TPM Use this setting when want to configure a set of registry variables in your database for the Tivoli Provisioning Manager.

WAS Use this setting when you want to configure a set of registry variables in your database for WebSphere® Application Server.

WC Use this setting when you want to configure a set of registry variables in your database for WebSphere Commerce.

WP Use this setting when you want to configure a set of registry variables in your database for WebSphere Portal.

Communications variables

You use communication variables to help control the flow of data on DB2 network connections and within the DB2 environment itself.

DB2CHECKCLIENTINTERVAL

- Operating system: All, server only
- Default=100, Values: A numeric value that is greater than or equal to zero.
- This variable specifies the frequency of TCP/IP client connection verifications during an active transaction. It permits early detection of client termination, instead of waiting until after the completion of the query. If this variable is set to 0, no verification is performed.

  Lower values cause more frequent checks. As a guide, for low frequency, use 100; for medium frequency, use 50; for high frequency use 10. The value is measured in an internal DB2 metric. The values represent a linear scale, that is, increasing the value from 50 to 100 doubles the interval. Checking more frequently for client status while executing a database request lengthens the time taken to complete queries. If the DB2 workload is heavy (that is, it involves many internal requests), setting **DB2CHECKCLIENTINTERVAL** to a low value has a greater impact on performance than in a situation where the workload is light.

DB2COMM

- Operating system: All, server only
- Default=NULL, Values: NPIPE, TCPIP, SSL
- This variable specifies the communication managers that are started when the database manager is started. If this variable is not set, no DB2 communications managers are started at the server. You can set this variable to any combination of the values, separated by commas.

  Note: The value NPIPE is valid only in Windows operating systems.

DB2FCMCOMM

- Operating system: All supported DB2 Enterprise Server Edition platforms
- Default=TCPIP4, Values: TCPIP4 or TCPIP6
- This variable specifies how the host names in the db2nodes.cfg file are resolved. All host names are resolved as IPv4 or IPv6. If an IP address instead of a host name is specified in db2nodes.cfg, the form of the IP determines if IPv4 or IPv6 is used. If **DB2FCMCOMM** is not set, its default setting of IPv4 means that only IPv4 hosts can be started.

  Note: If the IP format resolved from the hostname specified in db2nodes.cfg, or the IP format directly specified in db2nodes.cfg does not match the setting of **DB2FCMCOMM**, **db2start** will fail.

DB2_FORCE_NLS_CACHE

- Operating system: AIX, HP_UX, Solaris
- Default=FALSE, Values: TRUE or FALSE
- This variable is used to eliminate the chance of lock contention in multi-threaded applications. When this registry variable is TRUE, the code page and territory code information is saved the first time a thread accesses it. From that point, the cached information is used for any other thread that requests this information. This eliminates lock contention and

results in a performance benefit in certain situations. This setting should not be used if the application changes locale settings between connections. It is probably not needed in such a situation because multi-threaded applications typically do not change their locale settings because it is not *thread safe* to do so.

DB2_PMODEL_SETTINGS

- Operating system: All
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.
- This registry variable controls a set of parameters that allow you to modify the behavior of various aspects of the DB2 internal infrastructure. Separate parameters with a semicolon, as in the following example:

  ```
  db2set
  DB2_PMODEL_SETTINGS=MLN_REMOTE_LISTENER:TRUE;ENHANCED_ROLLBACK:TRUE;SRVLST_EQUAL_WEIGHT
  ```

- Parameters:

  ENHANCED_ROLLBACK

  - Default: FALSE
  - Values: TRUE, FALSE
  - Use the **ENHANCED_ROLLBACK** parameter to improve rollback behavior for units of work on DB2 servers in partitioned database environments. If you set this option to TRUE, rollback requests for units of work are sent only to logical database partitions that participated in the transaction.

  MLN_REMOTE_LISTENER

  - Default: FALSE
  - Values: TRUE, FALSE
  - Use the **MLN_REMOTE_LISTENER** parameter to start a TCP/IP listener on each logical database partition. If you set this option to TRUE, applications can connect directly to each logical database partition instead of routing requests through the database partition server that is assigned to logical port 0.

    If you set this option to TRUE, ensure that the additional TCP/IP listeners do not use ports that are required by other services.

  SRVLST_EQUAL_WEIGHT

  - Default: FALSE
  - Values: TRUE, FALSE
  - Use the **SRVLST_EQUAL_WEIGHT** parameter if you want non-zero member weights in the server list to always be identical, thereby overriding the default behavior in which member weights are computed based on load. Member weights as contained in the server list are used by a remote client to distribute workload when workload balancing (WLB) is enabled.

    If you set this option to TRUE, WLB at the client translates into even workload distribution among members with no regard for the member load.

DB2RSHCMD

- Operating system: UNIX, Linux

- Default=rsh (remsh on HP-UX), Values are a full path name to rsh, remsh, or ssh
- By default, DB2 database system uses rsh as the communication protocol when starting remote database partitions and with the **db2_all** script to run utilities and commands on all database partitions. For example, setting this registry variable to the full path name for ssh causes DB2 database products to use ssh as the communication protocol for the requested running of the utilities and commands. It may also be set to the full path name of a script that invokes the remote command program with appropriate default parameters. This variable is only required for partitioned databases, or for single-partition environments where the **db2start** command is run from a different server than where the DB2 product was installed and for rsh or ssh dependant utilities that have the capability of starting, stopping or monitoring a DB2 instance, such as **db2gcf**. The instance owner must be able to use the specified remote shell program to log in from each DB2 database node to each other DB2 database node, without being prompted for any additional verification or authentication (that is, passwords or password phrases).

For detailed instructions on setting the **DB2RSHCMD** registry variable to use a ssh shell with DB2, see the white paper "Configure DB2 Universal Database™ for UNIX to use OpenSSH."

DB2RSHTIMEOUT

- Operating system: UNIX, Linux
- Default=30 seconds, Values: 1 - 120
- This variable is only applicable if **DB2RSHCMD** is set to a non-null value. This registry variable is used to control the timeout period that the DB2 database system will wait for any remote command. After this timeout period, if no response is received, the assumption is made that the remote database partition is not reachable and the operation has failed.

  Note: The time value given is not the time required to run the remote command, it is the time needed to authenticate the request.

DB2SORCVBUF

- Operating system: All
- Default=65 536
- Specifies the value of TCP/IP receive buffers.

DB2SOSNDBUF

- Operating system: All
- Default=65 536
- Specifies the value of TCP/IP send buffers.

DB2TCP_CLIENT_CONTIMEOUT

- Operating system: All, client only
- Default=0 (no timeout), Values: 0 - 32 767 seconds
- The **DB2TCP_CLIENT_CONTIMEOUT** registry variable specifies the number of seconds a client waits for the completion on a TCP/IP connect operation. If a connection is not established in the seconds specified, then the DB2 database manager returns the error -30081 selectForConnectTimeout.

  There is no timeout if the registry variable is not set or is set to 0.

Note: Operating systems also have a connection timeout value that may take effect prior to the timeout you set using **DB2TCP_CLIENT_CONTIMEOUT**. For example, AIX has a default *tcp_keepinit*=150 (in half seconds) that would terminate the connection after 75 seconds.

- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2TCP_CLIENT_KEEPALIVE_TIMEOUT

- Operating system: AIX, HP-UX, Linux, Windows (client only)
- Default=15, Values: 0 - 32 767 seconds
- The **DB2TCP_CLIENT_KEEPALIVE_TIMEOUT** registry variable specifies the maximum time in seconds before an unresponsive TCP/IP client connection or attachment is detected as no longer alive. It is the client-side equivalent of **DB2TCP_SERVER_KEEPALIVE_TIMEOUT**. When this variable is not set, the default setting of 15 seconds is used. If set, this variable takes precedence over any keepAliveTimeout setting as specified in the db2dsdriver.cfg file.

  Changes to this variable take effect immediately for all future TCP/IP connections and attachments to the server

DB2TCP_CLIENT_RCVTIMEOUT

- Operating system: All, client only
- Default=0 (no timeout), Values: 0 - 32 767 seconds
- The **DB2TCP_CLIENT_RCVTIMEOUT** registry variable specifies the number of seconds a client waits for data on a TCP/IP receive operation. If data from the server is not received in the seconds specified, then the DB2 database manager returns the error -30081 selectForRecvTimeout.

  There is no timeout if the registry variable is not set or is set to 0.

  Note: The value of the **DB2TCP_CLIENT_RCVTIMEOUT** can be overridden by the CLI, using the db2cli.ini keyword **ReceiveTimeout** or the connection attribute SQL_ATTR_RECEIVE_TIMEOUT.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2TCPCONNMGRS

- Operating system: All
- Default=1 on serial machines; square root of the number of processors rounded up to a maximum of sixteen connection managers on symmetric multiprocessor machines. Values: 1 to 16
- The default number of connection managers is created if the registry variable is not set. If the registry variable is set, the value assigned here overrides the default value. The number of TCP/IP connection managers specified up to a maximum of 16 is created. If less than 1 is specified then **DB2TCPCONNMGRS** is set to a value of 1 and a warning is logged that the value is out of range. If greater than 16 is specified then **DB2TCPCONNMGRS** is set to a value of 16 and a warning is logged that the value is out of range. Values between 1 and 16 are used as given. When there is greater than one connection manager created, connection throughput should improve when multiple client connections are received simultaneously. There may be additional TCP/IP connection manager processes (on UNIX) or threads (on Windows operating

systems) if the user is running on a SMP machine, or has modified the **DB2TCPCONNMGRS** registry variable. Additional processes or threads require additional storage.

Note: Having the number of connection managers set to 1 causes a drop in performance on remote connections in systems with a lot of users, frequent connects and disconnects, or both.

DB2TCP_SERVER_KEEPALIVE_TIMEOUT

- Operating system: AIX, HP-UX, Linux, Windows (server only)
- Default=60, Values: 0 - 32 767 seconds
-

  The **DB2TCP_SERVER_KEEPALIVE_TIMEOUT** registry variable specifies the maximum time in seconds before an unresponsive TCP/IP client connection or attachment is detected as no longer alive. It is the server-side equivalent of **DB2TCP_CLIENT_KEEPALIVE_TIMEOUT** and keepAliveTimeout. When this variable is not set, the default setting of 60 seconds is used.

  Changes to this variable take effect immediately for all future TCP/IP connections and attachments to the server. There is no need to restart the server instance.

Command-line variables

You can set command line variables to control the default behavior of the command line in a DB2 product environment.

DB2BQTIME

- Operating system: All
- Default=1 second, Minimum value: 1 second
- This variable specifies the amount of time the command-line processor front end sleeps before it checks whether the back-end process is active and establishes a connection to it.

DB2BQTRY

- Operating system: All
- Default=60 retries, Minimum value: 0 retries
- This variable specifies the number of times the command-line processor front-end process tries to determine whether the back-end process is already active. It works in conjunction with **DB2BQTIME**.

DB2_CLP_EDITOR

- Operating system: All
- Default: Notepad(Windows), vi (UNIX), Values: Any valid editor that is located in the operating system path

  Note: This registry variable is not set to the default value during installation. Instead, the code that makes use of this variable uses a default value if the registry variable is not set.
- This variable determines the editor to be used when executing the **EDIT** command. From a CLP interactive session, the **EDIT** command launches an editor preloaded with a user-specified command which can then be edited and run.

DB2_CLP_HISTSIZE

- Operating system: All
- Default: 20, Values: 1-500 inclusive

  Note: This registry variable is not set to the default value during installation. Instead, the code that makes use of this variable uses a default value of 20 if the registry variable is not set or if it is set to a value outside of the valid range.
- This variable determines the number of commands stored in the command history during CLP interactive sessions. Because the command history is held in memory, a very high value for this variable might result in a performance impact depending on the number and length of commands run in a session.

DB2_CLPPROMPT

- Operating system: All
- Default=None (if it is not defined, "db2 => " will be used as the default CLP interactive prompt), Values: Any text string of length less than 100 that contains zero or more of the following tokens %i, %d, %ia, %da, or %n. Users need not set this variable unless they explicitly wish to change the default CLP interactive prompt (db2 =>).
- This registry variable allows a user to define the prompt to be used in the Command Line Processor (CLP) interactive mode. The variable can be set to any text string of length less than 100 characters containing zero or more of the optional tokens %i, %d, %ia, %da, or %n. When running in CLP interactive mode, the prompt to be used is constructed by taking the text-string specified in the **DB2_CLPPROMPT** registry variable and replacing all occurrences of the tokens %i, %d, %ia, %da, or %n by the local alias of the current attached instance, the local alias of the current database connection, the authorization ID of the current attached instance, the authorization ID of the current database connection, and newline (that is, a carriage-return) respectively.

  Note:

  1. If the **DB2_CLPPROMPT** registry variable is changed within CLP interactive mode, the new value for **DB2_CLPPROMPT** will not take effect until the CLP interactive mode has been closed and reopened.
  2. If no instance attachment exists, %ia is replaced by the empty string and %i is replaced by the value of the **DB2INSTANCE** registry variable. On Windows platforms only, if **DB2INSTANCE** is not set, %i is replaced by the value of the **DB2INSTDEF** registry variable. If neither of these variables are set, %i is replaced by the empty string.
  3. If no database connection exists, %da is replaced by the empty string and %d is replaced by the value of the **DB2DBDFT** registry variable. If the **DB2DBDFT** variable is not set, %d is replaced by the empty string.
  4. The interactive input prompt will always present the values for the authorization IDs, database names, and instance names in upper case.

DB2IQTIME

- Operating system: All
- Default=5 seconds, Minimum value: 1 second
- This variable specifies the amount of time the command line processor back end process waits on the input queue for the front end process to pass commands.

DB2RQTIME

- Operating system: All
- Default=5 seconds, Minimum value: 1 second
- This variable specifies the amount of time the command line processor back end process waits for a request from the front end process.

Partitioned database environment variables

You use partitioned database environment variables to control the default behavior of a partitioned database environment, including authorization, failover, and network behavior.

DB2CHGPWD_EEE

- Operating system: DB2 ESE on AIX, Linux, and Windows
- Default=NULL, Values: YES or NO
- This variable specifies whether you allow other users to change passwords on AIX or Windows ESE systems. You must ensure that the passwords for all database partitions or nodes are maintained centrally using either a Windows domain controller on Windows, or LDAP on AIX. If not maintained centrally, passwords may not be consistent across all database partitions or nodes. This could result in a password being changed only at the database partition to which the user connects to make the change.

DB2_FCM_SETTINGS

- Operating system: Linux
- Default=YES, Values:
  - FCM_MAXIMIZE_SET_SIZE:[YES|TRUE|NO|FALSE]. The default value for FCM_MAXIMIZE_SET_SIZE is YES.
  - FCM_CFG_BASE_AS_FLOOR: [YES|TRUE|NO|FALSE]. The default value for FCM_CFG_BASE_AS_FLOOR is NO.
- You can set the **DB2_FCM_SETTINGS** registry variable with the FCM_MAXIMIZE_SET_SIZE token to preallocate a default 4 GB of space for the fast communication manager (FCM) buffer. The token must have a value of either YES or TRUE to enable this feature.

  You can use the **DB2_FCM_SETTINGS** registry variable with the FCM_CFG_BASE_AS_FLOOR option to set the base value as the floor for the *fcm_num_buffers* and *fcm_num_channels* database manager configuration parameters. When the FCM_CFG_BASE_AS_FLOOR option is set to YES or TRUE, and these parameters are set to AUTOMATIC and have an initial or starting value, DB2 will not tune them below this value.

DB2_FORCE_OFFLINE_ADD_PARTITION

- Operating system: All
- Default=FALSE, Values: FALSE or TRUE
- This variable allows you to specify that add database partition server operations are to be performed offline. The default setting of FALSE indicates that DB2 database partition servers can be added without taking the database offline. However, if you want the operation to be performed offline or if some limitation prevents you from adding database partition servers when the database is online, set **DB2_FORCE_OFFLINE_ADD_PARTITION** to TRUE. When this variable is set to TRUE, new DB2 database partition servers are added according to the

Version 9.5 and earlier versions' behavior; that is, new database partition servers are not visible to the instance until it has been shut down and restarted.

DB2_NUM_FAILOVER_NODES

- Operating system: All
- Default=2, Values: 0 to the required number of database partitions
- Set **DB2_NUM_FAILOVER_NODES** to specify the number of additional database partitions that might need to be started on a machine in the event of failover.

  In a DB2 database high availability solution, if a database server fails, the database partitions on the failed machine can be restarted on another machine. The fast communication manager (FCM) uses **DB2_NUM_FAILOVER_NODES** to calculate how much memory to reserve on each machine to facilitate this failover.

  For example, consider the following configuration:

  – Machine A has two database partitions: 1 and 2.
  – Machine B has two database partitions: 3 and 4.
  – **DB2_NUM_FAILOVER_NODES** is set to 2 on both A and B.

  At START DBM, FCM will reserve enough memory on both A and B to manage up to four database partitions so that if one machine fails, the two database partitions on the failed machine can be restarted on the other machine. If machine A fails, database partitions 1 and 2 can be restarted on machine B. If machine B fails, database partitions 3 and 4 can be restarted on machine A.

DB2_PARTITIONEDLOAD_DEFAULT

- Operating system: All supported ESE platforms
- Default=YES, Values: YES or NO
- The **DB2_PARTITIONEDLOAD_DEFAULT** registry variable lets users change the default behavior of the load utility in an ESE environment when no ESE-specific load options are specified. The default value is `YES`, which specifies that in an ESE environment if you do not specify ESE-specific load options, loading is attempted on all database partitions on which the target table is defined. When the value is `NO`, loading is attempted only on the database partition to which the load utility is currently connected.

  Note: This variable is deprecated and may be removed in a later release. The LOAD command has various options that can be used to achieve the same behavior. You can achieve the same results as the `NO` setting for this variable by specifying the following with the **LOAD** command: `PARTITIONED DB CONFIG MODE LOAD_ONLY OUTPUT_DBPARTNUMS x`, where `x` is the partition number of the partition into which you want to load data.

DB2PORTRANGE

- Operating system: Windows
- Values: nnnn:nnnn
- This value is set to the TCP/IP port range used by FCM so that any additional database partitions created on another machine will also have the same port range.

DB2 pureScale environment variables

There are two environment variables that you can set for DB2 pureScale systems.

DB2_DATABASE_CF_MEMORY

- Operating system: All
- Default: 100, Values: -1, or 0 to 100
- Type: Float
- This variable is specifically for a DB2 pureScale environment.
- **DB2_DATABASE_CF_MEMORY** is used to indicate the proportion of the total CF memory (**CF_MEM_SZ**) that will be assigned to each database that has the **cf_db_mem_sz** database configuration parameter set to AUTOMATIC. Any database that has **cf_db_mem_sz** set to a specific value will ignore this registry variable.
- For all databases to have an equal share of the CF memory resources when more that 100 databases are active, the **DB2_DATABASE_CF_MEMORY** registry variable must use values less than 1. For example, if there are 200 active databases, each with an equal share of the CF memory, this registry variable must be set to 0.5.
- Use of the **DB2_DATABASE_CF_MEMORY** registry variable must be coordinated with the **cf_db_mem_sz** and **numdb** configuration parameters. For additional information, see DB2 pureScale CF memory parameter configuration.

DB2_MCR_RECOVERY_PARALLELISM_CAP

- Operating system: All
- This variable is specifically for a DB2 pureScale environment.
- In a multiple database environment, if member crash recovery (MCR) is required, the number of databases that will be recovered in parallel on each member is set by the value of the **numdb** configuration parameter or the **DB2_MCR_RECOVERY_PARALLELISM_CAP** registry variable, whichever value is smaller.

Query compiler variables

You use query compiler variables to control query compiler optimization decisions during DB2 query optimization process. These decisions include preventing specific optimization decisions and forcing a specific SQL query operation to occur.

DB2_ANTIJOIN

- Operating system: All
- Default=NO in a ESE environment, Default=EXTEND in a non-ESE environment, Values: YES, NO, or EXTEND
- When this variable is set to YES, the optimizer searches for opportunities to transform NOT EXISTS subqueries into anti-joins which can be processed more efficiently by DB2.

  When this variable is set to NO, the optimizer limits the opportunities to transform NOT EXISTS subqueries into anti-joins.

  When this variable is set to EXTEND, the optimizer searches for opportunities to transform both NOT IN and NOT EXISTS subqueries into anti-joins.

  Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_DEFERRED_PREPARE_SEMANTICS

- Operating system: All
- Default=NO, Values: YES or NO
- When set to YES, this registry variable enables deferred prepare semantics such that all untyped parameter markers used in PREPARE statements will derive their data types and length attributes based on the input descriptor associated with the subsequent OPEN or EXECUTE statements. This allows untyped parameter markers to be used in more places than was supported previously.

  Note: Setting **DB2_DEFERRED_PREPARE_SEMANTICS** to YES may cause unintended effects or results. In cases where the data type in the input descriptor is different from the data type derived using the rules for "Determining data types of untyped expressions," the following can occur:

  – The query performance is degraded because of the additional cast operation.
  – The query fails because a data type cannot be converted.
  – The query can return different results.

  For example, assume a table t1, with a column char_col which is defined as VARCHAR(10) with values '1', '100', '200', 'xxx'. A user runs the following query:

  ```
  select * from t1 where char_col = ?
  ```

  If the data type of the input parameter is INTEGER, and deferred prepare is being used, the column char_col is cast to numeric. However, the query fails because one of the rows in the table contains non-numeric data ('xxx') which cannot be converted to a numeric value.

  When set to YES_DBCS_GRAPHIC_TO_CHAR, this registry variable specifies that parameter markers are to be typed as VARCHAR instead of VARGRAPHIC. The **DB2_DEFERRED_PREPARE_SEMANTICS** registry variable has this setting implicitly if all of the following are true:

  – **DB2_DEFERRED_PREPARE_SEMANTICS** is not set (that is, set to NULL).
  – **The DB2_COMPATIBILITY_VECTOR** registry variable is set to ORA, MYS, or MSS.
  – You are in a double-byte character set (DBCS) environment.

  The **DB2_DEFERRED_PREPARE_SEMANTICS** registry variable must be set prior to issuing the **db2start** command.

  This registry variable is only recommended for Unicode and SBCS databases.

DB2_INLIST_TO_NLJN

- Operating system: All
- Default=NO, Values: YES or NO
- In some situations, the SQL and XQuery compiler can rewrite an IN list predicate to a join. For example, the following query:

  ```
  SELECT *
   FROM EMPLOYEE
   WHERE DEPTNO IN ('D11', 'D21', 'E21')
  ```

  could be written as:

```
SELECT *
 FROM EMPLOYEE, (VALUES 'D11', 'D21', 'E21') AS V(DNO)
 WHERE DEPTNO = V.DNO
```

This revision might provide better performance if there is an index on DEPTNO. The list of values would be accessed first and joined to EMPLOYEE with a nested loop join using the index to apply the join predicate.

Sometimes the optimizer does not have accurate information to determine the best join method for the rewritten version of the query. This can occur if the IN list contains parameter markers or host variables which prevent the optimizer from using catalog statistics to determine the selectivity. This registry variable causes the optimizer to favor nested loop joins to join the list of values, using the table that contributes the IN list as the inner table in the join.

Note: When either or both of the DB2 query compiler variables **DB2_MINIMIZE_LISTPREFETCH** and **DB2_INLIST_TO_NLJN**, are set to YES, they remain active even if REOPT(ONCE) is specified.

Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_LIKE_VARCHAR

- Operating system: All
- Default=Y,Y,
- Controls the use of sub-element statistics. These are statistics about the content of data in columns when the data has a structure in the form of a series of sub-fields or sub-elements delimited by blanks. Collection of sub-element statistics is optional and controlled by options in the **RUNSTATS** command or API.

  Important: This variable is deprecated and might be removed in a future release because you should only change the settings under the advisement of IBM service.

  This registry variable affects how the optimizer deals with a predicate of the form:

  ```
  COLUMN LIKE '%xxxxxx%'
  ```

  where the xxxxxx is any string of characters.

  The syntax showing how this registry variable is used is:

  ```
  db2set DB2_LIKE_VARCHAR=[Y|N|S|num1] [,Y|N|S|num2]
  ```

  where

  – The term preceding the comma, or the only term to the right of the predicate, means the following but only if the second term is specified as N or the column does not have positive sub-element statistics:
    - S – The optimizer estimates the length of each element in a series of elements concatenated together to form a column based on the length of the string enclosed in the % characters.
    - Y – The default. Use a default value of 1.9 for the algorithm parameter. Use a variable-length sub-element algorithm with the algorithm parameter.
    - N – Use a fixed-length sub-element algorithm.

- num1 – Use the value of num1 as the algorithm parameter with the variable length sub-element algorithm.

– The term following the comma means the following, but only for columns that do have positive sub-element statistics:
  - N – Do not use sub-element statistics. The first term takes effect
  - Y – The default. Use a variable-length sub-element algorithm that uses sub-element statistics together with the 1.9 default value for the algorithm parameter in the case of columns with positive sub-element statistics.
  - num2 – Use a variable-length sub-element algorithm that uses sub-element statistics together with the value of num2 as the algorithm parameter in the case of columns with positive sub-element statistics.

Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_MINIMIZE_LISTPREFETCH

- Operating system: All
- Default=NO, Values: YES or NO
- List prefetch is a special table access method that involves retrieving the qualifying RIDs from the index, sorting them by page number and then prefetching the data pages. Sometimes the optimizer does not have accurate information to determine if list prefetch is a good access method. This might occur when predicate selectivities contain parameter markers or host variables that prevent the optimizer from using catalog statistics to determine the selectivity.

  This registry variable prevents the optimizer from considering list prefetch in such situations.

  Note: When either or both of the DB2 query compiler variables **DB2_MINIMIZE_LISTPREFETCH** and **DB2_INLIST_TO_NLJN**, are set to YES, they remain active even if REOPT(ONCE) is specified.

  Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_NEW_CORR_SQ_FF

- Operating system: All
- Default=OFF, Values: ON or OFF
- Affects the selectivity value computed by the query optimizer for certain subquery predicates when it is set to ON. It can be used to improve the accuracy of the selectivity value of equality subquery predicates that use the MIN or MAX aggregate function in the SELECT list of the subquery. For example:

  ```
  SELECT * FROM T WHERE
  T.COL = (SELECT MIN(T.COL)
  FROM T WHERE ...)
  ```

  Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_OPT_MAX_TEMP_SIZE

- Operating system: All

- Default=NULL, Values: amount of space in megabytes that can be used by a query in all temporary table spaces
- Limits the amount of space that queries can use in the temporary table spaces. Setting **DB2_OPT_MAX_TEMP_SIZE** can cause the optimizer to choose a plan that is more expensive than would otherwise be chosen, but which uses less space in the temporary table spaces. If you set **DB2_OPT_MAX_TEMP_SIZE**, be sure to balance your need to limit use of temporary table space against the efficiency of the plan your setting causes to be chosen.

  If **DB2_WORKLOAD=SAP** is set, **DB2_OPT_MAX_TEMP_SIZE** is automatically set to `10 240` (10 GB).

  If you run a query that uses temporary table space in excess of the value set for **DB2_OPT_MAX_TEMP_SIZE**, the query does not fail, but you receive a warning that its performance may be suboptimal, as not all resources may be available.

  The operations considered by the optimizer that are affected by the limit set by **DB2_OPT_MAX_TEMP_SIZE** are:

  – Explicit sorts for operations such as ORDER BY, DISTINCT, GROUP BY, merge scan joins, and nested loop joins.
  – Explicit temporary tables
  – Implicit temporary tables for hash joins and duplicate merge joins

  Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_REDUCED_OPTIMIZATION

- Operating system: All
- Default=NO, Values: NO, YES, any integer, DISABLE, JUMPSCAN, NO_SORT_NLJOIN, or NO_SORT_MGJOIN
- This registry variable lets you request either reduced optimization features or rigid use of optimization features at the specified optimization level. If you reduce the number of optimization techniques that are used, you also reduce time and resource use during optimization.

  If you set this variable, the following syntax rules apply:

  – Separate each option with a comma (,), and ensure that no spaces appear before or after the comma.
  – Separate an option and the value for that option with a single space.
  – If the setting includes a space, enclose the setting in double quotation marks (“”).

  The following example shows the correct syntax:

  ```
  db2set DB2_REDUCED_OPTIMIZATION="NO_SORT_NLJOIN,JUMPSCAN ON"
  ```

  Note: Although optimization time and resource use might be reduced, the risk of producing a less than optimal data access plan is increased. Use this registry variable only when advised by IBM or one of its partners.

  – If set to NO

    The optimizer does not change its optimization techniques.

  – If set to YES

If the optimization level is 5 (the default) or lower, the optimizer disables some optimization techniques that might consume significant prepare time and resources but do not usually produce a better access plan.

If the optimization level is exactly 5, the optimizer scales back or disables some additional techniques, which might further reduce optimization time and resource use, but also further increase the risk of a less than optimal access plan. For optimization levels lower than 5, some of these techniques might not be in effect in any case. If they are, however, they remain in effect.

– If set to any integer

  The effect is the same as YES, with the following additional behavior for dynamically prepared queries optimized at level 5. If the total number of joins in any query block exceeds the setting, then the optimizer switches to greedy join enumeration instead of disabling additional optimization techniques as described previously for level 5 optimization levels. which implies that the query will be optimized at a level similar to optimization level 2.

– If set to DISABLE

  The behavior of the optimizer when unconstrained by this **DB2_REDUCED_OPTIMIZATION** variable is sometimes to dynamically reduce the optimization for dynamic queries at optimization level 5. This setting disables this behavior and requires the optimizer to perform full level 5 optimization.

– If set to JUMPSCAN

  Use this option to control if the DB2 optimizer can use jump scan operations. You can specify the following values:

  - OFF = The DB2 optimizer will not create plans using jumps cans.
  - ON = The DB2 optimizer uses cost-based analysis to determine whether to generate plans that use jump scans (default).

– If set to NO_SORT_NLJOIN

  The optimizer does not generate query plans that force sorts for nested loop joins (NLJN). These types of sorts can be useful for improving performance; therefore, be careful when using the NO_SORT_NLJOIN option, as performance can be severely impacted.

– If set to NO_SORT_MGJOIN

  The optimizer does not generate query plans that force sorts for merge scan joins (MSJN). These types of sorts can be useful for improving performance; therefore, be careful when using the NO_SORT_MGJOIN option, as performance can be severely impacted.

  Note that the dynamic optimization reduction at optimization level 5 takes precedence over the behavior described for optimization level of exactly 5 when **DB2_REDUCED_OPTIMIZATION** is set to YES as well as the behavior described for the integer setting.

– If set to ZZJN:

  Use this option to control how the DB2 optimizer uses the zigzag join method for star schema-based queries that contain one fact table. You can specify the following values:

  - OFF = The DB2 optimizer does not use the zigzag join method.

- ON = The DB2 optimizer uses cost-based analysis to determine whether to use the zigzag join method or a different join method (default).
- FORCE = If the zigzag join method is feasible, the DB2 optimizer uses the zigzag join method.

– If set to ZZJN_MULTI_FACT:

Use this option to control how the DB2 optimizer uses the zigzag join method for star schema-based queries that contain more than one fact table. You can specify the following values:

- OFF = The DB2 optimizer does not use the zigzag join method.
- ON = The DB2 optimizer uses cost-based analysis to determine whether to use the zigzag join method or a different join method (default).
- FORCE = If the zigzag join method is feasible, the DB2 optimizer uses the zigzag join method.

Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_SELECTIVITY

- Operating system: All
- Default=NO, Values: YES or NO
- This registry variable controls where the SELECTIVITY clause can be used in search conditions in SQL statements.

  When this registry variable is set to `NO`, the SELECTIVITY clause can only be specified in a user-defined predicate.

  When this registry variable is set to `YES`, the SELECTIVITY clause can be specified for the following predicates:

  – A user-defined predicate

  – A basic predicate in which at least one expression contains host variables or parameter markers

  Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_SQLROUTINE_PREPOPTS

- Operating system: All
- Default=Empty string, Values:
  – APREUSE {YES | NO}
  – BLOCKING {UNAMBIG | ALL | NO}
  – CONCURRENTACCESSRESOLUTION { USE CURRENTLY COMMITTED | WAIT FOR OUTCOME }
  – DATETIME {DEF | USA | EUR | ISO | JIS | LOC}
  – DEGREE {1 | *degree-of-parallelism* | ANY}
  – DYNAMICRULES {BIND | INVOKEBIND | DEFINEBIND | RUN | INVOKERUN | DEFINERUN}
  – EXPLAIN {NO | YES | ALL}
  – EXPLSNAP {NO | YES | ALL}
  – FEDERATED {NO | YES}
  – INSERT {DEF | BUF}

- ISOLATION {CS | RR | UR | RS | NC}
- OPTPROFILE {profile_name | schema_name.profile_name}
- QUERYOPT *optimization-level*
- REOPT {NONE | ONCE | ALWAYS}
- STATICREADONLY {YES|NO|INSENSITIVE}
- VALIDATE {RUN | BIND}

- The **DB2_SQLROUTINE_PREPOPTS** registry variable can be used to customize the precompile and bind options for SQL and XQuery procedures and functions. When setting this variable, separate each of the options with a space, as follows:

  ```
  db2set DB2_SQLROUTINE_PREPOPTS="BLOCKING ALL VALIDATE RUN"
  ```

  For a complete description of each option and its settings, see "BIND command."

If you want to achieve the same results as **DB2_SQLROUTINE_PREPOPTS** for select individual procedures, but without restarting the instance, use the SET_ROUTINE_OPTS procedure.

Performance variables

You can set these variables to improve the performance of DB2 products. Performance improvements that can be affected this way include access plan optimizations, specifying memory tuning operations and operating system resource policies.

DB2_ALLOCATION_SIZE

- Operating system: All
- Default=128 KB, Range: 64 KB - 256 MB
- Specifies the size of memory allocations for buffer pools.

  The potential advantage of setting a higher value for this registry variable is fewer allocations will be required to reach a desired amount of memory for a buffer pool.

  The potential cost of setting a higher value for this registry variable is wasted memory if the buffer pool is altered by a non-multiple of the allocation size. For example, if the value for **DB2_ALLOCATION_SIZE** is 8 MB and a buffer pool is reduced by 4 MB, this 4 MB will be wasted because an entire 8 MB segment cannot be freed.

Note: **DB2_ALLOCATION_SIZE** is deprecated and may be removed in a later release.

DB2_APM_PERFORMANCE

- Operating system: All
- Default=OFF, Values: ON or OFF
- Set this variable to ON to enable performance-related changes in the access plan manager (APM) that affect the behavior of the query cache (package cache). These settings are not usually recommended for production systems. They introduce some limitations, such as the possibility of out-of-package cache errors or increased memory use, or both.

  Setting **DB2_APM_PERFORMANCE** to ON also enables the NO PACKAGE LOCK mode. This mode allows the global query cache to operate without the use of package locks, which are internal system locks that protect cached

package entries from being removed. The NO PACKAGE LOCK mode might result in somewhat improved performance, but certain database operations are not allowed. These prohibited operations might include: operations that invalidate packages, operations that inoperate packages, and **PRECOMPILE**, **BIND**, and **REBIND**.

DB2ASSUMEUPDATE

- Operating system: All
- Default=OFF, Values: ON or OFF
- When enabled, this variable allows the DB2 database system to assume that all fixed-length columns provided in an UPDATE statement are being changed. This eliminates the need for the DB2 database system to compare the existing column values to the new values to determine if the column is actually changing. Using this registry variable can cause additional logging and index maintenance when columns are provided for update (for example, in a SET clause) but are not actually being modified.

  The activation of the **DB2ASSUMEUPDATE** registry variable is effective on the **db2start** command.

DB2_AVOID_PREFETCH

- Operating system: All
- Default=OFF, Values: ON or OFF
- Specifies whether prefetch should be used during crash recovery. If **DB2_AVOID_PREFETCH** =ON, prefetch is not used.

DB2_BACKUP_USE_DIO

- Operating system: All
- Default: OFF, Values: ON or OFF
- Specifies whether or not backup images are cached by the operating system. The default behavior is to cache the image file. When **DB2_BACKUP_USE_DIO** is set to ON, the backup image file is directly written to disk, bypassing the file cache.

  Setting this variable to ON might result in the operating system better utilizing memory resources because there is no benefit to caching the backup image file. This performance impact will have the largest benefit for Linux platforms. However, there may be a slight slowdown of the backup itself, so you should measure the change in backup performance when **DB2_BACKUP_USE_DIO** is set to ON.

  Note: Changing the value of this registry variable does not affect the behavior of the backup that is already running. Changing the value will take effect when the next backup is run, and it does not require an instance restart.

DB2BPVARS

- Operating system: As specified for each parameter
- Default=Path
- Two sets of parameters are available to tune buffer pools. One set of parameters, available only on Windows, specify that buffer pools should use scatter read for specific types of containers. The other set of parameters, available on all platforms, affect prefetching behavior.

Important: This performance variable has been deprecated in Version 9.5 and might be removed in a future release. For more information, see .

Parameters are specified in an ASCII file, one parameter on each line, in the form `parameter=value`. For example, a file named `bpvars.vars` might contain the following line:

```
NO_NT_SCATTER = 1
```

Assuming that `bpvars.vars` is stored in `F:\vars\`, to set these variables you execute the following command:

```
db2set DB2BPVARS=F:\vars\bpvars.vars
```

Scatter-read parameters

The scatter-read parameters are recommended for systems with a large amount of sequential prefetching against the respective type of containers and for which you have already set **DB2NTNOCACHE** to `ON`. These parameters, available only on Windows platforms, are `NT_SCATTER_DMSFILE`, `NT_SCATTER_DMSDEVICE`, and `NT_SCATTER_SMS`. Specify the`NO_NT_SCATTER` parameter to explicitly disallow scatter read for any container. Specific parameters are used to turn scatter read on for all containers of the indicated type. For each of these parameters, the default is `zero` (or `OFF`); and the possible values include: `zero` (or `OFF`) and 1 (or `ON`).

Note: You can turn on scatter read only if **DB2NTNOCACHE** is set to `ON` to turn Windows file caching off. If **DB2NTNOCACHE** is set to `OFF` or not set, a warning message is written to the administration notification log if you attempt to turn on scatter read for any container, and scatter read remains disabled.

DB2CHKPTR

- Operating system: All
- Default=`OFF`, Values: `ON` or `OFF`
- Specifies whether or not pointer checking for input is required.

DB2CHKSQLDA

- Operating system: All
- Default=`ON`, Values: `ON` or `OFF`
- Specifies whether or not SQLDA checking for input is required.

DB2_EVALUNCOMMITTED

- Operating system: All
- Default: `NO`, Values: `YES`, `NO`
- When enabled, this variable allows, where possible, scans to defer or avoid row locking until the data is known to satisfy predicate evaluation. With this variable enabled, predicate evaluation may occur on uncommitted data.

  DB2_EVALUNCOMMITTED is only applicable when currently committed semantics will not help avoid lock contentions. When this variable is set and currently committed is applicable to a scan, deleted rows will not be skipped and predicate evaluate will not occur on uncommitted data; the currently committed version of the rows and data will be processed instead.

  As well, **DB2_EVALUNCOMMITTED** is applicable only to statements using either Cursor Stability or Read Stability isolation levels. Furthermore,

deleted rows are skipped unconditionally on table scan access while deleted keys are not skipped for index scans unless the registry variable **DB2_SKIPDELETED** is also set.

The activation of the **DB2_EVALUNCOMMITTED** registry variable is effective on the **db2start** command. The decision as to whether deferred locking is applicable is made at statement compile or bind time.

DB2_EXTENDED_IO_FEATURES

- Operating system: AIX
- Default=OFF, Values: ON, OFF
- Set this variable to ON to enable features that enhance I/O performance. This enhancement includes improving the hit rate of memory caches as well as reducing the latency on high priority I/O. These features are only available on certain combinations of software and hardware configuration; setting this variable to ON for other configurations will be ignored by either the DB2 database management system or by the operating system. The minimum configuration requirements are:
  - Database version: DB2 V9.1
  - RAW device must be used for database containers (container on file systems is not supported)
  - Storage subsystem: Shark DS8000® supports all the enhanced I/O performance features. Refer to the Shark DS8000 documentation for setup and prerequisite information.

  The default I/O priority settings for HIGH, MEDIUM, and LOW are 3, 8, and 12, respectively; you can use the **DB2_IO_PRIORITY_SETTING** registry variable to change these settings.

DB2_EXTENDED_OPTIMIZATION

- Operating system: All
- Default: OFF, Values: ON, OFF, ENHANCED_MULTIPLE_DISTINCT, or IXOR
- This variable specifies whether or not the query optimizer uses optimization extensions to improve query performance. The ON, ENHANCED_MULTIPLE_DISTINCT, and IXOR values specify different optimization extensions. Use a comma-separated list when you want to use multiple values.

  The default behavior (specified by the OFF or IXOR values) is for the optimizer to extend the index ORing data access method to include OR predicates that reference any indexed column even when non-indexed column predicates are present. For example, consider the following two index definitions:

```
INDEX IX2:  dept    ASC
INDEX IX3:  job     ASC
```

  The following predicates can be satisfied by using these two indexes when the IXOR option is set:

```
WHERE
     dept = :hv1 OR
     (job = :hv2 AND
     years >= :hv3)
```

  In general, **DB2_EXTENDED_OPTIMIZATION** settings might not improve query performance in all environments. Testing should be done to determine individual query performance improvements.

  Important:

- The ENHANCED_MULTIPLE_DISTINCT and IXOR values have been deprecated in Version 10.1 and might be removed in a future release. Removing ENHANCED_MULTIPLE_DISTINCT makes new enhancements that improve the performance of multiple distinct queries available. The IXOR value is redundant because it specifies the default behavior. For more details, see "Registry variables with changed behaviors" in *What's New for DB2 Version 10.1*.
- The ENHANCED_MULTIPLE_DISTINCT value will only take effect dynamically if it was enabled when the instance was last started.

DB2_IO_PRIORITY_SETTING

- Operating system: AIX
- Values: HIGH:#,MEDIUM:#,LOW:#, where # can be 1 to 15
- This variable is used in combination with the **DB2_EXTENDED_IO_FEATURES** registry variable. This registry variable provides a means to override the default HIGH, MEDIUM, and LOW I/O priority settings for the DB2 database system, which are 3, 8, and 12, respectively. This registry variable must be set prior to the start of an instance; any modification requires an instance restart. Note that setting this registry variable alone does not enable the enhanced I/O features, **DB2_EXTENDED_IO_FEATURES** must be set to enable them. All system requirements for **DB2_EXTENDED_IO_FEATURES** also apply to this registry variable.

DB2_KEEP_AS_AND_DMS_CONTAINERS_OPEN

- Operating system: All
- Default: NO, Values: YES or NO
- When you set this variable to ON, each DMS table space container has a file handle opened until the database is deactivated. Query performance might improve because the overhead to open the containers is eliminated. You should use this registry only in pure DMS environments, otherwise performance of queries against SMS table spaces might be impacted negatively.

DB2_KEEPTABLELOCK

- Operating system: All
- Default: OFF, Values: ON, TRANSACTION, OFF, CONNECTION
- When this variable is set to ON or TRANSACTION, this variable allows the DB2 database system to maintain the table lock when an Uncommitted Read or Cursor Stability isolation level is closed. The table lock that is kept is released at the end of the transaction, just as it would be released for Read Stability and Repeatable Read scans.

  When this variable is set to CONNECTION, a table lock is released for an application until the application either rolls back the transaction or the connection is reset. The table lock continues to be held across commits and application requests to drop the table lock are ignored by the database. The table lock remains allocated to the application. Thus, when the application re-requests the table lock, the lock is already available.

  For application workloads that can leverage this optimization, performance should improve. However, the workloads of other application executing concurrently might be impacted. Other applications might get blocked from accessing a given table resulting in poor concurrency. DB2 SQL catalog tables are not impacted by this setting. The CONNECTION setting also includes the behavior described with the ON or TRANSACTION setting.

This registry variable is checked at statement compile or bind time.

DB2_LARGE_PAGE_MEM

- Operating system: AIX, Linux, Windows Server 2003
- Default=NULL, Values: Use * to denote all applicable memory regions that should use large page memory, or a comma-separated list of specific memory regions that should use large page memory. Available regions vary by operating system. On AIX, the following regions can be specified: DB, DBMS, FCM, APPL, or PRIVATE. On Linux, the following region can be specified: DB. On Windows Server 2003, the following region can be specified: DB. Huge page memory is only available on AIX.
- The **DB2_LARGE_PAGE_MEM** registry variable is used to enable large page or huge page support. Setting **DB2_LARGE_PAGE_MEM**=DB enables large-page memory for the database shared memory region, and if **database_memory** is set to AUTOMATIC, disables automatic tuning of this shared memory region by STMM. On AIX, setting DB2_LARGE_PAGE_MEM=DB:16GB enables huge page memory for the database shared memory region.

  Memory access-intensive applications that use large amounts of virtual memory may obtain performance improvements by using large or huge pages. To enable the DB2 database system to use them, you must first configure the operating system to use large or huge pages.

  To enable large pages for agent private memory on 64-bit DB2 for AIX (the **DB2_LARGE_PAGE_MEM**=PRIVATE setting), you have to configure large pages on the operating system and the instance owner must possess the CAP_BYPASS_RAC_VMM and CAP_PROPAGATE capabilities.

  On AIX 5L™, you can set this variable to FCM. FCM memory resides in its own memory set, so you must add the FCM keyword to the value of the **DB2_LARGE_PAGE_MEM** registry variable to enable large pages for FCM memory.

  On Linux, there is an additional requirement for the availability of the *libcap.so.1* library. This library must be installed for this option to work. If this option is turned on and the library is not on the system, the DB2 database disables the large kernel pages and continues to function as it would without them.

  On Linux, to verify that large kernel pages are available, issue the following command:

  ```
  cat /proc/meminfo
  ```

  If large kernel pages are available, the following three lines should appear (with different numbers depending on the amount of memory configured on your server):

  ```
  HugePages_Total:   200
  HugePages_Free:    200
  Hugepagesize:    16384 kB
  ```

  If you do not see these lines, or if the HugePages_Total is 0, you need to configure the operating system or kernel.

  On Windows, the amount of large page memory that is available on the system is less than the total available memory. After the system has been running for some time, memory can become fragmented, and the amount of large page memory decreases. The **DB2_ALLOCATION_SIZE** registry variable should be set to a high value, such as 256 MB, in order to achieve consistent performance allocating large memory pages on Windows. (Note that **DB2_ALLOCATION_SIZE** requires you to stop and restart the instance for changes to take effect.)

DB2_LOGGER_NON_BUFFERED_IO

- Operating system: All
- Default=AUTOMATIC, Values: AUTOMATIC, ON, or OFF
- This variable allows you to control whether direct I/O (DIO) will be used on the log file system. When **DB2_LOGGER_NON_BUFFERED_IO** is set to AUTOMATIC, active log windows (namely, the primary log files) will be opened with DIO, and all other logger files will be buffered. When it is set to ON, all log file handles will be opened with DIO. When it is set to OFF, all log files handles will be buffered.

DB2MAXFSCRSEARCH

- Operating system: All
- Default=5, Values: -1, 1 to 33 554
- Specifies the number of free space control record (FSCRs) to search when adding a record to a table. The default is to search five FSCRs. Modifying this value allows you to balance insert speed with space reuse. Use large values to optimize for space reuse. Use small values to optimize for insert speed. Setting the value to -1 forces the database manager to search all FSCRs.

DB2_MAX_INACT_STMTS

- Operating system: All
- Default=Not set, Values: up to 4 000 000 000
- This variable overrides the default limit on the number of inactive statements kept by any one application. You can choose a different value in order to increase or reduce the amount of system monitor heap used for inactive statement information. The default limit is 250.

  The system monitor heap can become exhausted if an application contains a very high number of statements in a unit of work, or if there are a large number of applications executing concurrently.

DB2_MAX_NON_TABLE_LOCKS

- Operating system: All
- Default=YES, Values: See description
- This variable defines the maximum number of NON table locks a transaction can have before it releases all of these locks. NON table locks are table locks that are kept in the hash table and transaction chain even when the transaction has finished using them. Because transactions often access the same table more than once, retaining locks and changing their state to NON can improve performance.

  For best results, the recommended value for this variable is the maximum number of tables expected to be accessed by any connection. If no user-defined value is specified, the default value is as follows: If the **locklist** size is greater than or equal to

  SQLP_THRESHOLD_VAL_OF_LRG_LOCKLIST_SZ_FOR_MAX_NON_LOCKS

  (currently 8000), the default value is

  SQLP_DEFAULT_MAX_NON_TABLE_LOCKS_LARGE

  (currently 150). Otherwise, the default value is

  SQLP_DEFAULT_MAX_NON_TABLE_LOCKS_SMALL

  (currently 0).

DB2_MDC_ROLLOUT

- Operating system: All
- Default=IMMEDIATE, Values: IMMEDIATE, OFF, or DEFER
- This variable enables a performance enhancement known as "rollout" for deletions from MDC tables. Rollout is a faster way of deleting rows in an MDC table, when entire cells (intersections of dimension values) are deleted in a search DELETE statement. The benefits are reduced logging and more efficient processing.
- There are three possible outcomes of the variable setting:
  - No rollout - if OFF is specified
  - Immediate rollout - if IMMEDIATE is specified.
  - Rollout with deferred index cleanup - if DEFER is specified
- If the value is changed after startup, any new compilations of a statement will respect the new registry value setting. For statements that are in the package cache, no change in delete processing will be made until the statement is recompiled. The SET CURRENT MDC ROLLOUT MODE statement overrides the value of **DB2_MDC_ROLLOUT** at the application connection level.
- In DB2 Version 9.7 and later releases, the DEFER value is not supported for data partitioned MDC table with partitioned RID indexes. Only the OFF and IMMEDIATE values 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.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2MEMDISCLAIM

- Operating system: ALL
- Default=YES, Values: YES or NO
- Memory used by DB2 database system processes might have some associated paging space. This paging space might remain reserved even when the associated memory has been freed. Whether or not this is so depends on the operating system's (tunable) virtual memory management allocation policy. The **DB2MEMDISCLAIM** registry variable controls whether DB2 agents explicitly request that the operating system disassociate the reserved paging space from the freed memory.

  A **DB2MEMDISCLAIM** setting of YES results in smaller paging space requirements, and possibly less disk activity from paging. A **DB2MEMDISCLAIM** setting of NO results in larger paging space requirements, and possibly more disk activity from paging. In some situations, such as if paging space is plentiful and real memory is so plentiful that paging never occurs, a setting of NO provides a minor performance improvement.

DB2_MEM_TUNING_RANGE

- Operating system: All
- Default=NULL, Values: a sequence of percentages *n*, *m* where *n=minfree* and *m=maxfree*

- The amount of physical memory that the DB2 instance leaves free is important because this dictates how much memory other applications running on the same machine are able to use. When self tuning of database shared memory is enabled, the amount of physical memory left free by a given instance depends on the need for memory by the instance (and its active databases). When an instance is in urgent need of additional memory, it will allocate memory until the free physical memory on the system reaches the percentage specified by *minfree*. When the instance is less in need of memory, it will maintain a larger amount of free physical memory, specified as a percentage by *maxfree*. As a result, it is a requirement that the value set for *minfree* must be less than the value of *maxfree*.

  If this variable is not set, the DB2 database manager will calculate values for *minfree* and *maxfree* based on the amount of memory on the server. The setting of this variable has no effect unless you are running the self-tuning memory manager (STMM) and have **database_memory** set to AUTOMATIC.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2_MMAP_READ

- Operating system: AIX
- Default=OFF, Values: ON or OFF
- This variable is used in conjunction with **DB2_MMAP_WRITE** to allow the DB2 database system to use mmap as an alternate method of I/O.

  When these variables are set to ON, data is read into and written from the DB2 buffer pools using memory mapped I/O, and subsequently removed from the file system cache. This avoids double-caching of DB2 data. However, the recommended method to bypass the file system cache is to specify the NO FILE SYSTEM CACHING clause at the table space level, and to leave these variables at the default setting of OFF.

DB2_MMAP_WRITE

- Operating system: AIX
- Default=OFF, Values: ON or OFF
- This variable is used in conjunction with **DB2_MMAP_READ** to allow the DB2 database system to use mmap as an alternate method of I/O.

  When these variables are set to ON, data is read into and written from the DB2 buffer pools using memory mapped I/O, and subsequently removed from the file system cache. This avoids double-caching of DB2 data. However, the recommended method to bypass the file system cache is to specify the NO FILE SYSTEM CACHING clause at the table space level, and to leave these variables at the default setting of OFF.

DB2_NO_FORK_CHECK

- Operating system: UNIX
- Default=OFF, Values: ON or OFF
- When this variable is enabled, the DB2 runtime client minimizes checks to determine if the current process is a result of a fork call. This can improve performance of DB2 applications that do not use the fork() api.

DB2NTMEMSIZE

- Operating system: Windows

- Default= (varies by memory segment)
- Windows requires that all shared memory segments be reserved at DLL initialization time in order to guarantee matching addresses across processes. **DB2NTMEMSIZE** permits the user to override the DB2 defaults on Windows if necessary. In most situations, the default values should be sufficient. The memory segments, default sizes, and override options are:
  1. Parallel FCM Buffers: default size is 512 MB on 32-bit platforms, 4.5 GB on 64-bit platforms; override option is FCM:*number_of_bytes*
  2. Fenced Mode Communication: default size is 80 MB on 32-bit platforms, 512 MB on 64-bit platforms; override option is APLD:*number_of_bytes*
  3. Message Query Memory: default size is 4 MB on 32-bit and 64-bit platforms; override option is QUE:<number of bytes>.

  More than one segment may be overridden by separating the override options with a semicolon (;). For example, on a 32-bit version of DB2, to limit the FCM buffers to 1 GB, and the fenced stored procedures limit to 256 MB, use:

  ```
  db2set DB2NTMEMSIZE=FCM:1073741824;APLD:268435456
  ```

  To increase the message queue memory to 64 MB, use:

  ```
  db2set DB2NTMEMSIZE=QUE:67108864
  ```

DB2NTNOCACHE

- Operating system: Windows
- Default=OFF, Values: ON or OFF
- The **DB2NTNOCACHE** registry variable specifies whether the DB2 database system opens database files with a NOCACHE option. If **DB2NTNOCACHE** is set to ON, file system caching is eliminated. If **DB2NTNOCACHE** is set to OFF, the operating system caches DB2 files. This applies to all data except for files that contain long fields or LOBs. Eliminating system caching allows more memory to be available to the database so that the buffer pool or sort heap can be increased.

  In Windows, files are cached when they are opened, which is the default behavior. One MB is reserved from a system pool for every 1 GB in the file. Use this registry variable to override the undocumented 192 MB limit for the cache. When the cache limit is reached, an out-of-resource error is given.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

Note: For table space containers, using the NO FILE SYSTEM CACHING clause with the ALTER TABLESPACE or CREATE TABLESPACE statement reports the same benefit as setting **DB2NTNOCACHE** to ON.

DB2NTPRICLASS

- Operating system: Windows
- Default=NULL, Values: R, H, (any other value)
- Sets the priority class for the DB2 instance (program DB2SYSCS.EXE). There are three priority classes:
  - NORMAL_PRIORITY_CLASS (the default priority class)
  - REALTIME_PRIORITY_CLASS (set by using R)
  - HIGH_PRIORITY_CLASS (set by using H)

This variable is used in conjunction with individual thread priorities (set using **DB2PRIORITIES**) to determine the absolute priority of DB2 threads relative to other threads in the system.

Note: **DB2NTPRICLASS** is deprecated and should only be used at the recommendation of service. Use DB2 service classes to adjust agent priority and prefetch priority. Care should be taken when using this variable. Misuse could adversely affect overall system performance.

For more information, please refer to the SetPriorityClass() API in the Win32 documentation.

DB2NTWORKSET

- Operating system: Windows
- Default=1,1
- Used to modify the minimum and maximum working-set size available to the DB2 database manager. By default, when Windows is not in a paging situation, the working set of a process can grow as large as needed. However, when paging occurs, the maximum working set that a process can have is approximately 1 MB. **DB2NTWORKSET** allows you to override this default behavior.

  Specify **DB2NTWORKSET** using the syntax **DB2NTWORKSET**=*min*,*max*, where *min* and *max* are expressed in megabytes.

DB2_OVERRIDE_BPF

- Operating system: All
- Default=Not set, Values: a positive numeric number of pages OR <entry>[;<entry>...] where <entry>=<buffer pool ID>,<number of pages>
- This variable specifies the size of the buffer pool, in pages, to be created at database activation, rollforward recovery, or crash recovery. It is useful when memory constraints cause failures to occur during database activation, rollforward recovery, or crash recovery. The memory constraint could arise either in the rare case of a real memory shortage or, because of the attempt by the database manager to allocate a large buffer pool, in the case where there were inaccurately configured buffer pools. For example, when even a minimal buffer pool of 16 pages is not brought up by the database manager, try specifying a smaller number of pages using this environment variable. The value given to this variable overrides the current buffer pool size.

  You can also use <entry>[;<entry>...] where <entry>=<buffer pool ID>,<number of pages> to temporarily change the size of all or a subset of the buffer pools so that they can start up.

DB2_PINNED_BP

- Operating system: AIX, HP-UX, Linux
- Default=NO, Values: YES or NO
- Setting this variable to YES causes DB2 to request that the Operating System pins DB2's Database Shared Memory. When configuring DB2 to pin Database Shared Memory, care should be taken to ensure that the system is not overcommitted, as the operating system will have reduced flexibility in managing memory.

  On Linux, in addition to modifying this registry variable, the library, libcap.so.1 is also required.

Setting this variable to YES means that self tuning for database shared memory (activated by setting the **database_memory** configuration parameter to AUTOMATIC) cannot be enabled.

For HP-UX in a 64-bit environment, in addition to modifying this registry variable, the DB2 instance group must be given the MLOCK privilege. To do this, a user with root access rights performs the following actions:

1. Adds the DB2 instance group to the /etc/privgroup file. For example, if the DB2 instance group belongs to db2iadm1 group then the following line must be added to the /etc/privgroup file:

   ```
   db2iadm1 MLOCK
   ```

2. Issues the following command:

   ```
   setprivgrp -f /etc/privgroup
   ```

DB2PRIORITIES

- Operating system: All
- Values setting is platform dependent
- Controls the priorities of DB2 processes and threads.

Note: DB2PRIORITIES is deprecated and should only be used at the recommendation of service. Use DB2 service classes to adjust agent priority and prefetch priority.

DB2_RCT_FEATURES

- Operating system: All
- Default: NULL. Values: GROUPUPDATE=[ON|OFF]. The default value for GROUPUPDATE is OFF.
- This variable allows for optimized and reduced update processing for a searched UPDATE statement which targets multiple rows in an range clustered table when only equal predicates on the leading and subset of key sequence columns are specified. Logging is also reduced due to a single log record for all rows updated on a page, instead of a log record for each row updated.

  Usage :

  ```
  db2set DB2_RCT_FEATURES=GROUPUPDATE=ON
  ```

DB2_RESOURCE_POLICY

- Operating system: AIX 5 or higher, all Linux except zSeries (32–bit), Windows Server 2003 or higher
- Default=Not set, Values: valid path to configuration file, AUTOMATIC on POWER7® systems running AIX 6.1 Technology Level (TL) 5 or higher
- Defines a resource policy which can be used to limit what operating system resources are used by the DB2 database or it contains rules for assigning specific operating system resources to specific DB2 database objects. For example, on AIX, Linux, or Windows operating systems, this registry variable can be used to limit the set of processors that the DB2 database system uses. The extent of resource control varies depending on the operating system.

  On AIX NUMA and Linux NUMA enabled machines, a policy can be defined which specifies what resource sets the DB2 database system uses. When resource set binding is used, each individual DB2 process is bound to a particular resource set. This can be beneficial in some performance tuning scenarios.

On POWER7 systems running AIX 6.1 Technology Level (TL) 5 or higher this variable can be set to AUTOMATIC. With this setting the DB2 database system automatically determines the hardware topology and assigns engine dispatchable units (EDUs) to the various hardware modules in such a way that memory can be more efficiently shared between multiple EDUs that need to access the same regions of memory. This setting is intended for larger POWER7 systems with 16 or more cores and can result in enhanced query performance on some workloads. It is best to run a performance analysis of the workload before and after setting this variable to AUTOMATIC to validate any performance improvements.

You can set the registry variable to indicate the path to a configuration file which defines a policy for binding DB2 processes to operating system resources. The resource policy allows you to specify a set of operating system resources to restrict the DB2 database system. Each DB2 process is bound to a single resource of the set. Resource assignment occurs in a circular round robin fashion.

Sample configuration files:

Example 1: Bind all DB2 processes to either CPU 1 or 3.

```
<RESOURCE_POLICY>
  <GLOBAL_RESOURCE_POLICY>
   <METHOD>CPU</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>1</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>3</RESOURCE>
   </RESOURCE_BINDING>
  </GLOBAL_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

Example 2: (AIX only) Bind DB2 processes to one of the following resource sets: sys/node.03.00000, sys/node.03.00001, sys/node.03.00002, sys/node.03.00003

```
<RESOURCE_POLICY>
  <GLOBAL_RESOURCE_POLICY>
   <METHOD>RSET</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00000</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00001</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00002</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.03.00003</RESOURCE>
   </RESOURCE_BINDING>
  </GLOBAL_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

Note: For AIX only, use of the RSET method requires CAP_NUMA_ATTACH capability.

Example 3: (Linux only) Bind all memory from bufferpool IDs 2 and 3 which are associated with the SAMPLE database to NUMA node 3. Also use 80 percent of the total database memory for the binding to NUMA node 3 and leave 20 percent to be striped across all nodes for non-bufferpool specific memory.

```
<RESOURCE_POLICY>
  <DATABASE_RESOURCE_POLICY>
   <DBNAME>sample</DBNAME>
   <METHOD>NODEMASK</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>3</RESOURCE>
    <DBMEM_PERCENTAGE>80</DBMEM_PERCENTAGE>
    <BUFFERPOOL_BINDING>
     <BUFFERPOOL_ID>2</BUFFERPOOL_ID>
     <BUFFERPOOL_ID>3</BUFFERPOOL_ID>
    </BUFFERPOOL_BINDING>
   </RESOURCE_BINDING>
  </DATABASE_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

Example 4: (For Linux and Windows only) Define two distinct processor sets specified by CPU masks 0x0F and 0xF0. Bind DB2 processes and bufferpool ID 2 to processor set 0x0F and DB2 processes and bufferpool ID 3 to processor set 0xF0. For each processor set, use 50 percent of the total database memory for the binding.

This resource policy is useful when a mapping between processors and NUMA nodes is desired. An example of such a scenario is a system with 8 processors and 2 NUMA nodes where processors 0 to 3 belong to NUMA node 0 and processors 4 to 7 belong to NUMA node 1. This resource policy allows for processor binding while implicitly maintaining memory locality (that is, a hybrid of CPU method and NODEMASK method).

```
 <RESOURECE_POLICY>
   <DATABASE_RESOURCE_POLICY>
     <DBNAME>sample</DBNAME>
     <METHOD>CPUMASK</METHOD>
     <RESOURCE_BINDING>
       <RESOURCE>0x0F</RESOURCE>
       <DBMEM_PERCENTAGE>50</DBMEM_PERCENTAGE>
       <BUFFERPOOL_BINDING>
         <BUFFERPOOL_ID>2</BUFFERPOOL_ID>
       </BUFFERPOOL_BINDING>
     </RESOURCE_BINDING>
     <RESOURCE_BINDING>
       <RESOURCE>0x0F</RESOURCE>
       <DBMEM_PERCENTAGE>50</DBMEM_PERCENTAGE>
       <BUFFERPOOL_BINDING>
         <BUFFERPOOL_ID>3</BUFFERPOOL_ID>
       </BUFFERPOOL_BINDING>
     </RESOURCE_BINDING>
   </DATABASE_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

Note: Use of the RSET method requires CAP_NUMA_ATTACH capability and is not supported on Linux.

The configuration file specified by the **DB2_RESOURCE_POLICY** registry variable accepts a SCHEDULING_POLICY element. You can use the SCHEDULING_POLICY element on some platforms to select

– The operating system scheduling policy used by the DB2 server

  You can set an operating system scheduling policy for DB2 on AIX, and for DB2 on Windows using the **DB2NTPRICLASS** registry variable.

– The operating system priorities used by individual DB2 server agents

Alternatively, you can use the registry variables **DB2PRIORITIES** and **DB2NTPRICLASS** to control the operating system scheduling policy and set DB2 agent priorities. However, the specification of a

SCHEDULING_POLICY element in the resource policy configuration file provides a single place to specify both the scheduling policy and the associated agent priorities.

Example 1: Selection of the AIX SCHED_FIFO2 scheduling policy with a priority boost for the DB2 log writer and reader processes.

```
<RESOURCE_POLICY>
 <SCHEDULING_POLICY>
  <POLICY_TYPE>SCHED_FIFO2</POLICY_TYPE>
  <PRIORITY_VALUE>60</PRIORITY_VALUE>

  <EDU_PRIORITY>
   <EDU_NAME>db2loggr</EDU_NAME>
   <PRIORITY_VALUE>56</PRIORITY_VALUE>
  </EDU_PRIORITY>

  <EDU_PRIORITY>
   <EDU_NAME>db2loggw</EDU_NAME>
   <PRIORITY_VALUE>56</PRIORITY_VALUE>
  </EDU_PRIORITY>
 </SCHEDULING_POLICY>
</RESOURCE_POLICY>
```

Example 2: Replacement for DB2NTPRICLASS=H on Windows.

```
<RESOURCE_POLICY>
 <SCHEDULING_POLICY>
  <POLICY_TYPE>HIGH_PRIORITY_CLASS</POLICY_TYPE>
 </SCHEDULING_POLICY>
</RESOURCE_POLICY>
```

DB2_SELUDI_COMM_BUFFER

- Operating system: All
- Default=OFF, Values: ON or OFF
- This variable is used during the processing of blocking cursors over SELECT from UPDATE, INSERT, or DELETE (UDI) queries. When enabled, this registry variable prevents the result of a query from being stored in a temporary table. Instead, during the OPEN processing of a blocking cursor for a SELECT from UDI query, the DB2 database system attempts to buffer the entire result of the query directly into the communications buffer memory area.

  Note: If the communications buffer space is not large enough to hold the entire result of query, an SQLCODE -906 error is issued, and the transaction is rolled back. See the **aslheapsz** and **rqrioblk** database manager configuration parameters for information on adjusting the size of the communication buffer memory area for local and remote applications respectively.

  This registry variable is not supported when intrapartition parallelism is enabled.

  Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_SET_MAX_CONTAINER_SIZE

- Operating system: All
- Default=Not set, Values: -1, any positive integer greater than 65 536 bytes

- This registry variable allows you to limit the size of individual containers for automatic storage table spaces with the AutoResize feature enabled.

  Note: Although you can specify **DB2_SET_MAX_CONTAINER_SIZE** in bytes, kilobytes, or megabytes, **db2set** indicates its value in bytes.
- If the value is set to -1, there will be no limit to the size of a container.

DB2_SKIPDELETED

- Operating system: All
- Default=OFF, Values: ON or OFF
- When enabled, this variable allows statements using either Cursor Stability or Read Stability isolation levels to unconditionally skip deleted keys during index access and deleted rows during table access. With **DB2_EVALUNCOMMITTED** enabled, deleted rows are automatically skipped, but uncommitted pseudo-deleted keys in indexes are not skipped unless **DB2_SKIPDELETED** is also enabled.

  DB2_SKIPDELETED is only applicable when currently committed semantics will not help avoid lock contentions. When this variable is set and currently committed is applicable to a scan, deleted rows will not be skipped; their currently committed version will be processed instead

  This registry variable does not impact the behavior of cursors on the DB2 catalog tables.

  This registry variable is activated with the **db2start** command.

DB2_SKIPINSERTED

- Operating system: All
- Default=OFF, Values: ON or OFF
- When the **DB2_SKIPINSERTED** registry variable is enabled, it allows statements using either Cursor Stability or Read Stability isolation levels to skip uncommitted inserted rows as if they had not been inserted. This registry variable does not impact the behavior of cursors on the DB2 catalog tables. This registry variable is activated at database startup, while the decision to skip uncommitted inserted rows is made at statement compile or bind time.

  This registry variable has no effect if currently committed semantics are being used. That is, even if **DB2_SKIPINSERTED** is set to OFF and currently committed behavior is enabled, uncommitted inserted rows are still skipped.

  Note: Skip inserted behavior is not compatible with tables that have pending rollout cleanup. As a result, scanners might wait for locks on a RID only to discover that the RID is part of a rolled out block.

DB2_SMS_TRUNC_TMPTABLE_THRESH

v98_u3

- Operating system: All
- Default=-2, Values: -2, -1, 0to *n*, where *n*=the number of extents per temporary table in the SMS table space container that are to be maintained
- This variable specifies a minimum file size threshold at which the file representing a temporary table is maintained in SMS table spaces.

The default setting for this variable is -2, which means that there will not be any unnecessary file system access for any spilled SMS temporary objects whose size is less than or equal to 1 extent * number of containers. Temporary objects that are larger than this are truncated to 0 extent.

When this variable is set to 0, no special threshold handling is done. Instead, once a temporary table is no longer needed, that file is truncated to 0 extent. When the value of this variable is greater than 0, a larger file is maintained. Objects larger than the threshold will be truncated to the threshold size. This reduces some of the system overhead involved in dropping and recreating the file each time a temporary table is used.

If this variable is set to -1, the file is not truncated and the file is allowed to grow indefinitely, restricted only by system resources.

DB2_SORT_AFTER_TQ

- Operating system: All
- Default=NO, Values: YES or NO
- Specifies how the optimizer works with directed table queues in a partitioned database environment when the receiving end requires the data to be sorted and the number of receiving nodes is equal to the number of sending nodes.

  When **DB2_SORT_AFTER_TQ**=NO, the optimizer tends to sort at the sending end and merge the rows at the receiving end.

  When **DB2_SORT_AFTER_TQ**=YES, the optimizer tends to transmit the rows unsorted, not merge at the receiving end, and sort the rows at the receiving end after receiving all the rows.

  Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_SQLWORKSPACE_CACHE

- Operating system: All
- Default: 30, Values: 10 - 2000
- This variable allows you to control the amount of caching of previously used sections in the SQL Workspace.

  The SQL Workspace contains allocations, in the form of sections, for the execution of SQL. Each SQL statement (static or dynamic) that is being executed on behalf of an application must maintain a unique copy of the section in the SQL Workspace for the duration of execution of that statement. Once the execution of the statement is complete, the section becomes inactive and the memory allocations associated with an inactive section can either be freed, or they can remain cached in the SQL Workspace. When a new execution of the same SQL statement occurs from any connection, it may find a cached copy of the section in the SQL Workspace left from a previous execution, thus saving the costs associated with allocating and initializing a new copy of the section. In such a manner, the SQL Workspace contains both active sections–corresponding to currently executing SQL–and cached sections that are not currently executing.

  The value for this registry variable specifies the percentage of memory allocations that are allowed to remain cached in the SQL Workspace. This caching is expressed as a percentage of the memory allocations for active sections. Thus, for example, a value of 50 would mean that the

SQL workspace contains all of the active (currently executing) sections and up to 50% more of previously executed cached sections that can be reused. You would adjust the setting for **DB2_SQLWORKSPACE_CACHE** based on how much of the SQL workspace you want to make available for reuse. For example, increasing the size of this variable, can result in some performance improvements for OLTP workloads. On the other hand, a higher setting also means that there is an increase in the size of the application shared heap.

Note: if the **appl_memory** database configuration parameter is not set to AUTOMATIC, the size of the SQL Workspace may also be limited by the **appl_memory** and the SQL Workspace may not provide as much caching as the **DB2_SQLWORKSPACE_CACHE** setting might allow for; you might want to consider increasing **appl_memory** (or setting it to AUTOMATIC) in such a case.
This registry variable is not dynamic

DB2_TRUSTED_BINDIN

- Operating system: All
- Default=OFF, Values: OFF, ON, or CHECK
- When **DB2_TRUSTED_BINDIN** is enabled, it speeds up the execution of query statements containing host variables within an embedded unfenced stored procedure.

  When this variable is enabled, there is no conversion from the external SQLDA format to an internal DB2 format during the binding of SQL and XQuery statements contained within an embedded unfenced stored procedure. This will speed up the processing of the embedded SQL and XQuery statements.

  The following data types are not supported in embedded unfenced stored procedures when this variable is enabled:
  - SQL_TYP_DATE
  - SQL_TYP_TIME
  - SQL_TYP_STAMP
  - SQL_TYP_CGSTR
  - SQL_TYP_BLOB
  - SQL_TYP_CLOB
  - SQL_TYP_DBCLOB
  - SQL_TYP_CSTR
  - SQL_TYP_LSTR
  - SQL_TYP_BLOB_LOCATOR
  - SQL_TYP_CLOB_LOCATOR
  - SQL_TYP_DCLOB_LOCATOR
  - SQL_TYP_BLOB_FILE
  - SQL_TYP_CLOB_FILE
  - SQL_TYP_DCLOB_FILE
  - SQL_TYP_BLOB_FILE_OBSOLETE
  - SQL_TYP_CLOB_FILE_OBSOLETE
  - SQL_TYP_DCLOB_FILE_OBSOLETE

  If these data types are encountered, an SQLCODE -804, SQLSTATE 07002 error is returned.

Note: The data type and length of the input host variable must match the internal data type and length of the corresponding element exactly. For host variables, this requirement will always be met. However, for parameter markers, care must be taken to ensure that matching data types are used. The CHECK option can be used to ensure that the data types and lengths match for all input host variables, but this option negates most of the performance improvements.

Note: **DB2_TRUSTED_BINDIN** is deprecated and will be removed in a later release.

DB2_USE_ALTERNATE_PAGE_CLEANING

- Operating system: All
- Default=Not set, Values: ON or OFF
- This variable specifies whether a DB2 database uses the alternate method of page cleaning algorithms or the default method of page cleaning. When this variable is set to ON, the DB2 system writes changed pages to disk, keeping ahead of LSN_GAP and proactively finding victims. Doing this allows the page cleaners to better utilize available disk I/O bandwidth. When this variable is set to ON, the *chngpgs_thresh* database configuration parameter is no longer relevant because it does not control page cleaner activity.

DB2_USE_FAST_PREALLOCATION

- Operating system: AIX, Linux and Solaris on VeritasVxFS, JFS2, GPFS, ext4 (Linux only) file systems
- Default: ON for Veritas VxFS, JFS2, GPFS, and ext4, Values: ON or OFF
- Allows the fast preallocation file system feature to reserve table space, and speed up the process of creating or altering large table spaces and database restore operations. This speed improvement is implemented at a small delta cost of performing actual space allocation during runtime when rows are inserted.

  To disable fast preallocation, set **DB2_USE_FAST_PREALLOCATION** to OFF. This might improve runtime performance, at the cost of slower table space creation and database restore times, on some operating systems, especially AIX, when there is a large volume of inserts and selects on same table space. Note that once fast preallocation is disabled, the database has to be restored.

DB2_USE_IOCP

- Operating system: AIX
- Default=ON, Values: ON or OFF
- This variables enables the use of AIX I/O completion ports (IOCP) when submitting and collecting asynchronous I/O (AIO) requests. This feature is used to enhance performance in a non-uniform memory access (NUMA) environment by avoiding remote memory access.

Miscellaneous variables

You can set other DB2 variables, including those that control the default administration server, client path, and the ability to commit changes made to DB2 data when you exit an application.

DB2ADMINSERVER

- Operating system: Windows and UNIX

- Default: NULL
- Specifies the DB2 Administration Server.

DB2_ATS_ENABLE

- Operating system: All
- Default: NULL, Values: YES/TRUE/ON/1 or NO/FALSE/OFF/0
- This variable controls whether the administrative task scheduler is running. The administrative task scheduler is disabled by default. When the scheduler is disabled, you can use the built-in procedures and views to define and modify tasks but the scheduler will not execute the tasks.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2AUTH

- Operating system: All
- Default: Not set. Values: DISABLE_CHGPASS, OSAUTHDB, SQLADM_NO_RUNSTATS_REORG, TRUSTEDCLIENT_DATAENC, TRUSTEDCLIENT_SRVRENC
- This variable allows you to tune the behavior of user authentication. Valid values are as follows:
  - ALLOW_LOCAL_FALLBACK: This value allows the DB2 server to fall back to using SERVER authentication for local implicit connects or attaches when the server is configured to use Generic Security Service (GSS) plugins. When ALLOW_LOCAL_FALLBACK is enabled, for local implicit connects, the userid and password plugin specified by the **srvcon_pw_plugin**database manager configuration parameter is used for authenticating the user, instead of using the specified GSS authentication, such as KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN, or GSS_SERVER_ENCRYPT.

    A local implicit connect is created when you issue a connect to a local database (note that *local* means only IPC, not TCP/IP) without providing a userid or password. DB2 uses the userid of the current session or process for the userid of the connect. The password plugins provided by DB2 assume that a userid retrieved from the operating system has already been authenticated by the operating system and, therefore, password validation is not necessary.

    Note: If you provide a userid and password, it is not considered a local implicit connect and the ALLOW_LOCAL_FALLBACK option does not apply.

    The password plugin DB2 uses is determined by the **srvcon_pw_plugin** database manager configuration parameter. If the **srvcon_pw_plugin** parameter is set to IBMLDAPauthserver, the IBMLDAPauthserver plugin processes the local implicit connect. If the **srvcon_pw_plugin** parameter is set to a custom security plugin, the custom plugin processes the local implicit connect. If the **srvcon_pw_plugin** parameter is not set, the default plugin (IBMOSauthserver) processes the local implicit connect. The DB2-provided security plugins always allow a local implicit connect because they assume that the user has been validated by the OS.
  - DISABLE_CHGPASS: This value disables the ability to change the password from the client.

– OSAUTHDB: This value instructs the DB2 database manager to use the authentication and group setting for a user on the AIX operating system. Transparent LDAP support has also been extended to the Linux, HP-UX and Solaris operating systems. The LDAP server can be any one of the following:
  - IBM Tivoli Directory Server (ITDS)
  - Microsoft Active Directory (MSAD)
  - Sun One Directory Server
– SQLADM_NO_RUNSTATS_REORG: This value, introduced in DB2 Version 9.7 Fix Pack 5, disables the ability of users with SQLADM authority to perform a reorg or runstats operation.
– TRUSTEDCLIENT_DATAENC: This value forces untrusted clients to use DATA_ENCRYPT. This value is not applicable to a DB2 Connect gateway.
– TRUSTEDCLIENT_SRVRENC: This value forces untrusted clients to use SERVER_ENCRYPT. This value is not applicable to a DB2 Connect gateway.

DB2_BCKP_INCLUDE_LOGS_WARNING

- Operating system: All
- Default: FALSE, Values: FALSE, TRUE
- Specifies whether online backups which fail to include all of the necessary log files should still be allowed to complete successfully. By default, online backups that do not explicitly specify either the INCLUDE LOGS or the EXCLUDE LOGS option fail if all of the logs are not successfully included. When this variable is set to TRUE, these backups will be allowed to succeed with a warning (SQL2440W).

  In SAP environments, when **DB2_WORKLOAD**=SAP is set, the default value of this registry variable is TRUE.

DB2_BCKP_PAGE_VALIDATION

- Operating system: All
- Default: FALSE, Values: FALSE, TRUE
- Specifies whether DMS and AS page validation occurs during a backup.

DB2CLIINIPATH

- Operating system: All
- Default: NULL
- Used to override the default path of the CLI/ODBC configuration file (db2cli.ini) and specify a different location on the client. The value specified must be a valid path on the client system.

DB2_COMMIT_ON_EXIT

- Operating system: UNIX
- Default: OFF, Values: OFF/NO/0 or ON/YES/1
- On UNIX operating systems, prior to DB2 UDB Version 8, DB2 committed any remaining in-flight transactions on successful application exit. In DB2 UDB Version 8, the behavior was changed so that in-flight transactions were rolled back on exit. This registry variable allows users with embedded SQL applications which depend on the earlier behavior to continue to enable it in DB2 Version 9. This registry variable does not affect JDBC, CLI, and ODBC applications.

  Note that this registry variable is deprecated, and the commit-on-exit behavior will no longer be supported in future release. Users should

determine whether any of their applications developed prior to DB2 Version 9 continue to depend on this functionality, and add the appropriate explicit COMMIT or ROLLBACK statements to the application as required. If the registry variable is turned on, care should be taken not to implement new applications which fail to explicitly COMMIT before exit.

Most users should leave this registry variable at the default setting.

DB2_COMMON_APP_DATA_PATH

- Operating system: Windows
- Default: Windows' common application data path.
  - For Windows XP and Windows 2003 operating systems: `C:\Documents and Settings\All Users\Application Data\`
  - For Windows Vista and later operating systems: `C:\ProgramData\`
- Points to user-defined location that holds the DB2 common application data for the DB2 copy. This registry variable is populated if **DB2_COMMON_APP_DATA_TOP_PATH** is specified during the response file installation or if "DB2 Common Application Data Top Path" field is populated during the custom installation step.

  Starting in Version 9.7 Fix Pack 5, this registry variable is visible in **db2set** command output but is not changeable. Any attempts to change given registry value will result in DBI1301E Invalid value error.

DB2_COMPATIBILITY_VECTOR

- Operating system: All
- Default: `NULL`, Values: `NULL` or `00` to `FFF`
- The **DB2_COMPATIBILITY_VECTOR** registry variable is used to enable one or more DB2 compatibility features introduced since DB2 Version 9.5. These features ease the task of migrating applications written for other relational database vendors to DB2 Version 9.5 or later.
- **DB2_COMPATIBILITY_VECTOR** is represented as a hexadecimal value, and each bit in the variable enables one of the DB2 compatibility features as outlined in the DB2_COMPATIBILITY_VECTOR values table. To enable all of the supported compatibility features, set the registry variable to the value ORA (which is equivalent to the hexadecimal value FFF). This is the recommended setting.

DB2CONNECT_DISCONNECT_ON_INTERRUPT

- Operating system: All
- Default: `NO`, Values: `YES/TRUE/1` or `NO/FALSE/0`
- When set to `YES` (`TRUE` or `1`), this variable specifies that the connection to a Version 8 (or higher) DB2 Universal Database z/OS server should be broken immediately when an interrupt occurs. You can use this variable in the following configurations:
  - If you are running a DB2 client with a Version 8 (or higher) DB2 UDB z/OS server, set **DB2CONNECT_DISCONNECT_ON_INTERRUPT** to `YES` on the client.
  - If you are running a DB2 client through a DB2 Connect gateway to a Version 8 (or higher) DB2 UDB z/OS server, set **DB2CONNECT_DISCONNECT_ON_INTERRUPT** to `YES` on the gateway.

DB2_CREATE_DB_ON_PATH

- Operating system: Windows

- Default: NULL, Values: YES or NO
- Set this registry variable to YES to enable support for the use of a path (as well as a drive) as a database path. The setting of **DB2_CREATE_DB_ON_PATH** is checked when a database is created, when the database manager configuration parameter **dftdbpath** is set, and when a database is restored. The fully qualified database path can be up to 215 characters in length.

  If **DB2_CREATE_DB_ON_PATH** is not set (or is set to NO) and you specify a path for the database path when creating or restoring a database, error SQL1052N is returned.

  If **DB2_CREATE_DB_ON_PATH** is not set (or is set to NO) and you update the **dftdbpath** database manager configuration parameter, error SQL5136N is returned.

  CAUTION:
  If path support is used to create new databases, applications written prior to DB2 Version 9.1 using the db2DbDirGetNextEntry() API or an older version of it, might not work correctly. Please refer to http://www.ibm.com/software/data/db2/support/db2_9/ for details on various scenarios and the proper course of action.

DB2_DDL_SOFT_INVAL

- Operating system: All
- Default: ON, Values: ON or OFF
- Enables soft invalidation of applicable database objects when they are dropped or altered.

  When **DB2_DDL_SOFT_INVAL** is set to ON, any DDL operation, such as drop, alter, or detach, can start without waiting for transactions referencing the same objects to finish. Current® executions dependant on the objects will continue with the original object definition, while new executions will utilize the changed object. This allows for better concurrency when issuing DDL statements.

  Note: The new soft invalidation capabilities only apply to dynamic packages. Any objects with static packages will still require a hard invalidation.

DB2_DISABLE_FLUSH_LOG

- Operating system: All
- Default: OFF, Values: ON or OFF
- Specifies whether to disable closing the active log file when the online backup is completed.

  When an online backup completes, the last active log file is truncated, closed, and made available to be archived. This ensures that your online backup has a complete set of archived logs available for recovery. You might want to disable closing the last active log file if you are concerned that you are wasting portions of the Log Sequence Number (LSN) space. Each time an active log file is truncated, the LSN is incremented by an amount proportional to the space truncated. If you perform a large number of online backups each day, you might disable closing the last active log file.

  You might also want to disable closing the last active log file if you find you are receiving log full messages a short time after the completion of the online backup. When a log file is truncated, the reserved active log space is incremented by the amount proportional to the size of the

truncated log. The active log space is freed once the truncated log file is reclaimed. The reclamation occurs a short time after the log file becomes inactive. During the short interval between these two events, you may receive log full messages.

During any backup which includes logs, this registry variable is ignored, since the active log file must be truncated and closed in order for the backup to include the logs.

- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2_DISPATCHER_PEEKTIMEOUT

- Operating system: All
- Default: 1, Values: 0 to 32767 seconds; 0 denotes that timeout is immediate
- **DB2_DISPATCHER_PEEKTIMEOUT** allows you to adjust the time, in seconds, that a dispatcher waits for a client's connection request before handing the client off to an agent. In most cases, you should not need to adjust this registry variable. This registry variable only affects instances that have DB2 Connect connection concentrator enabled.

  This registry variable and the **DB2_SERVER_CONTIMEOUT** registry variable both configure the handling of a new client during connect time. If there are many slow clients connecting to an instance, the dispatcher may be held up for up to 1 second to timeout each client, causing the dispatcher to become a bottle neck, if many clients are connecting simultaneously. If an instance with multiple active databases is experiencing very slow connection times, **DB2_DISPATCHER_PEEKTIMEOUT** may be lowered to 0. Lowering **DB2_DISPATCHER_PEEKTIMEOUT** causes the dispatcher to only look into the client's connect request if it is already there; the dispatcher will not wait for the connect request to arrive. If an invalid value is set, the default value is used. This registry variable is not dynamic.

DB2_DJ_INI

- Operating system: All
- Default:
  - UNIX: *db2_instance_directory*/cfg/db2dj.ini
  - Windows: *db2_install_directory*\cfg\db2dj.ini
- Specifies the absolute path name of the federation configuration file, for example: db2set DB2_DJ_INI=$HOME/sqllib/cfg/my_db2dj.ini This file contains the settings for data source environment variables. These environment variables are used by the Informix® wrapper and by the wrappers provided by InfoSphere® Federation Server.

  Here is a sample federation configuration file:

```
INFORMIXDIR=/informix/client_sdk
INFORMIXSERVER=inf93
ORACLE_HOME=/usr/oracle9i
SYBASE=/sybase/V12
SYBASE_OCS=OCS-12_5
```

  The following restrictions apply to the db2dj.ini file:

  - Entries must follow the format *evname=value* where *evname* is the name of the environment variable and *value* is its value.
  - The environment variable name has a maximum length of 255 bytes.
  - The environment variable value has a maximum length of 765 bytes.

This variable is ignored unless the database manager parameter **federated** is set to YES.

DB2_DMU_DEFAULT

- Operating system: All
- Default: NULL, Values: : IMPLICITLYHIDDENMISSING, IMPLICITLYHIDDENINCLUDE
- This variable allows you to set the default behavior of whether implicitly hidden columns are included when the column list is omitted by the load, import, ingest, and export utilities. Valid values are as follows:

  NULL
  It means that no default behavior is specified. If the table has implicitly hidden columns, the column list must be explicitly specified or the hidden column options must be specified by the utilities. Otherwise an error occurs.

  IMPLICITLYHIDDENMISSING
  The utilities assume that the implicitly hidden columns are not included by default unless the column list or the hidden column options are specified.

  IMPLICITLYHIDDENINCLUDE
  The utilities assume that the implicitly hidden columns are included by default, when neither the column list nor the hidden column options are specified.

  Consider the following examples of how the setting for **DB2_DMU_DEFAULT** affects the result of a load operation:

  – **DB2_DMU_DEFAULT** is set as IMPLICITLYHIDDENMISSING

  ```
  db2 load from delfile1 of del insert into table1
  ```

  If table1 has implicitly hidden columns, the load utility assumes that the data for implicitly hidden columns is not in the input file.

  – **DB2_DMU_DEFAULT** is set as IMPLICITLYHIDDENINCLUDE

  ```
  db2 load from delfile1 of del insert into table1
  ```

  If table1 has implicitly hidden columns, the load utility assumes that the data for implicitly hidden columns is in the input file and attempts to load it.

DB2_DOCHOST

- Operating system: All
- Default: Not set (but DB2 will still try to access the Information Center from the IBM website), Values: http://*hostname* where *hostname*= valid host name or IP address
- Specifies the host name on which the *DB2 Information Center* is installed. This variable can be automatically set during the installation of the *DB2 Information Center* if the automatic configuration option is selected in the DB2 Setup wizard.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2_DOCPORT

- Operating system: All

- Default: NULL, Values: any valid port number
- Specifies the port number through which the DB2 help system serves the DB2 documentation. This variable can be automatically set during the installation of the *DB2 Information Center* if the automatic configuration option is selected in the DB2 Setup wizard.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2DSDRIVER_CFG_PATH

- Operating system: All
- Default: NULL
- Used to override the default directory of the db2dsdriver.cfg configuration file.
- In Version 10.1 Fix Pack 2 and later fix packs, multiple directories for the db2dsdriver.cfg configuration file can be specified with use of the delimiter character.
  - On the Windows operating systems, the delimiter character is a semicolon (;).
  - On Linux and UNIX operating systems, the delimiter character is either a semicolon (;) or a colon (:). The semicolon character and the colon character cannot be used together as a delimiter for the **DB2DSDRIVER_CFG_PATH** variable value.

  Do not use the delimiter character in the directory name. The directories are searched sequentially in order specified in the **DB2DSDRIVER_CFG_PATH** variable value.
- The period (.) specifies the current directory.
- For more information about the db2dsdriver.cfg configuration file, see the Related reference section.

DB2DSDRIVER_CLIENT_HOSTNAME

- Operating system: All
- Default: NULL
- Used to override the default client host name of the (db2dsdriver.cfg) configuration file. This variable forces CLI to pick the client host name entry from the automatic client reroute section of db2dsdriver.cfg file.

DB2_ENABLE_AUTOCONFIG_DEFAULT

- Operating system: All
- Default: NULL, Values: YES or NO
- This variable controls whether the Configuration Advisor is run automatically at database creation. If **DB2_ENABLE_AUTOCONFIG_DEFAULT** is not set (null), the effect is the same as if the variable was set to YES and the Configuration Advisor is run at database creation. You do not need to restart the instance after you set this variable. If you execute the **AUTOCONFIGURE** command or run **CREATE DB AUTOCONFIGURE**, these commands override the setting of **DB2_ENABLE_AUTOCONFIG_DEFAULT**.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2_ENABLE_LDAP

- Operating system: All

- Default: NO, Values: YES or NO
- Specifies whether or not the Lightweight Directory Access Protocol (LDAP) is used. LDAP is an access method to directory services.

DB2_EVMON_EVENT_LIST_SIZE

- Operating system: All
- Default: 0 (no limit), Values: A value specified in KB/Kb/kb, MB/Mb/mb, or GB/Gb/gb; While there is no fixed upper limit for this variable, it is limited by the amount of available memory from the monitor heap.
- This registry variable specifies the maximum number of bytes that can be queued up waiting to be written to a particular event monitor. Once this limit is reached, agents attempting to send event monitor records will wait until the queue size drops below this threshold.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

Note: If activity records cannot be allocated from the monitor heap, they will be dropped. To prevent this from happening, set the **mon_heap_sz** configuration parameter to AUTOMATIC. If you have **mon_heap_sz** set to a specific value, ensure that **DB2_EVMON_EVENT_LIST_SIZE** is set to a smaller value. These actions, however, cannot guarantee that activity records will not be dropped, as the monitor heap is also used for tracking other monitor elements.

DB2_EVMON_STMT_FILTER

- Operating system: All
- Default: Not set; Values:
  - ALL: Indicates that the output for all statement event monitors is to be filtered. This option is exclusive.
  - '*nameA nameB nameC*': Where each name in the string represents the name of an event monitor for which records are to be filtered. If more than one name is supplied, each name must be separated by a single blank. All input names will be made uppercase by DB2. The maximum number of event monitors you can specify is 32. Each monitor name can be up to a maximum of 18 characters long.
  - '*nameA:op1,op2 nameB:op1,op2 nameC:op1*': Where each name in the string represents the name of an event monitor for which records are to be filtered. Each option (*op1*, *op2*, etc) represents an integer value mapping to a specific SQL operation. Specifying integer values allows users to determine which rules to apply to which event monitor.
- **DB2_EVMON_STMT_FILTER** can be used to reduce the number of records written by a statement event monitor. When set, this registry variable causes only the records for the following SQL operations to be written to the specified event monitor:

Table 123. Values to use for **DB2_EVMON_STMT_FILTER** *to restrict event monitor output to specific SQL operations*

SQL operation	Integer value mapping
SELECT	15
EXECUTE	2
EXECUTE_IMMEDIATE	3

Table 123. Values to use for **DB2_EVMON_STMT_FILTER** *to restrict event monitor output to specific SQL operations (continued)*

SQL operation	Integer value mapping
CLOSE	6
STATIC COMMIT	8
STATIC ROLLBACK	9
CALL	12
PRE_EXEC	17

All other operations will not appear in the output of the statement event monitor. To customize the set of operations for which records are written to the event monitor, use integer values.

Example 1:

```
db2set DB2_EVMON_STMT_FILTER= 'mon1 monitor3'
```

In this example, mon1 and monitor3 event monitors will receive a record for a restricted list of application requests. For example, if an application being monitored by the mon1 statement event monitor prepares a dynamic SQL statement, opens a cursor based on that statement, fetches 10,000 rows from that cursor, and then issues a cursor close request, only a record for a close request will appear in the mon1 event monitor output.

Example 2:

```
db2set DB2_EVMON_STMT_FILTER='evmon1:3,8 evmon2:9,15
```

In this example, evmon1 and evmon2 will receive a record for a restricted list of application requests. For example, if an application being monitored by the evmon1 statement event monitor issues a create statement, only the execute immediate and static commit operations will appear in the evmon1 event monitor output. If an application being monitored by the evmon2 statement event monitor performs SQL involving both a select and a static rollback only these two operations will appear in the evmon2 event monitor output.

Note: Refer to the `sqlmon.h` header file for definitions of database system monitor constants.

DB2_EXTSECURITY

- Operating system: Windows
- Default: `YES`, Values: `YES` or `NO`
- Prevents unauthorized access to DB2 by locking by locking DB2 objects (system files, directories, and IPC objects). To avoid potential problems, this registry variable should not be turned off. If **DB2_EXTSECURITY** is not set, its value is interpreted as `YES` on DB2 database sever products and `NO` on clients.

DB2_FALLBACK

- Operating system: Windows
- Default: `OFF`, Values: `ON` or `OFF`
- This variable allows you to force all database connections off during the fallback processing. It is used in conjunction with the failover support in the Windows environment with Microsoft Cluster Server (MSCS). If

DB2_FALLBACK is not set or is set to OFF, and a database connection exists during the fall back, the DB2 resource cannot be brought offline. This will mean the fallback processing will fail.

DB2_FMP_COMM_HEAPSZ

- Operating system: Windows, UNIX
- Default: 20 MB, or enough space to run 10 fenced routines (whichever is larger). On AIX, the default is 256 MB
- This variable specifies, in 4 KB pages, the size of the pool used for fenced routine invocations, such as stored procedure or user-defined function calls. The space used by each fenced routine is twice the value of the **aslheapsz** configuration parameter.

  If you are running a large number of fenced routines on your system, you may need to increase the value of this variable. If you are running a very small number of fenced routines, you can reduce it.

  Setting this value to 0 means that no set is created, and as a result no fenced routines can be invoked. It also means that the health monitor and the automatic database maintenance functionality (such as automatic backups, statistics collection, and **REORG**) will be disabled since this functionality relies on the fenced routine infrastructure.

  If you are running SAS in-database analytics (enabled by setting the **DB2_SAS_SETTINGS** registry variable), the memory for connections to the SAS embedded process (EP) is also allocated from the FMP heap. Guidelines for fenced routines apply when the heap is adjusted to accommodate connections running queries that include in-database analytics. As a general rule, you can expect the FMP heap memory requirements to increase by 120 KB. If, however, you specify the COMM_BUFFER_SZ option for the **DB2_SAS_SETTINGS** registry variable, the FMP heap memory requirements increase by twice the value of the COMM_BUFFER_SZ option multiplied by the number of concurrent SAS queries that you want to support.

DB2_GRP_LOOKUP

- Operating system: Windows
- Default: NULL, Values: LOCAL, DOMAIN, TOKEN, TOKENLOCAL, TOKENDOMAIN
- This variable specifies which Windows security mechanism is used to enumerate the groups to which a user belongs.

DB2_HADR_BUF_SIZE

- Operating system: All
- Default: 2***logbufsz**
- This variable specifies the standby log receiving buffer size in unit of log pages. If not set, DB2 will use two times the primary **logbufsz** configuration parameter value for the standby receiving buffer size. The maximum size that can be specified is 4 GB. This variable should be set in the standby instance. It is ignored by the primary database.

  If HADR synchronization mode (the **hadr_syncmode** database configuration parameter) is set to ASYNC, during peer state, a slow standby might cause the send operation on the primary to stall and therefore block transaction processing on the primary. A larger than default log-receiving buffer can be configured on a standby database to allow it to hold more unprocessed log data. This may allow for brief periods where the primary generates log data faster than the standby can consume it, without blocking transaction processing at the primary.

Note: A larger log receiving buffer size can help absorb peak transaction loads on the primary database, but the buffer will still fill up if the average replay rate on the standby database is slower than the log rate on the primary database.

DB2_HADR_NO_IP_CHECK

- Operating system: All
- Default: OFF, Values: ON | OFF
- Specifies whether to bypass IP check for HADR connections
- This variable is primarily used in Network Address Translation (NAT) environments to bypass IP cross check for HADR connections. Use of this variable is not recommended in other environments because it weakens the sanity check of the HADR configuration. By default, configuration consistency for the local and remote host parameters is verified when an HADR connection is established. Hostnames are mapped to IP addresses for the cross check. Two checks are performed:
  - **HADR_LOCAL_HOST** parameter on primary = **HADR_REMOTE_HOST** parameter on standby
  - **HADR_REMOTE_HOST** parameter on primary = **HADR_LOCAL_HOST** parameter on standby

  The connection will be closed if the check fails.

  When this parameter is turned on, no IP check occurs.

DB2_HADR_PEER_WAIT_LIMIT

- Operating system: All
- Default: **0** (meaning no limit) Values: 0 to max unsigned 32 bit integer, inclusive
- When the **DB2_HADR_PEER_WAIT_LIMIT** registry variable is set, the HADR primary database will break 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 will break the connection to the standby database. If the peer window is disabled, the primary database will enter disconnected state and logging resumes. If the peer window is enabled, the primary database will enter 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 once 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.

  This parameter has no effect on a standby database, but it is recommended that the same value be used on both primary and standby databases. Invalid values (not a number or negative numbers) will be interpreted as 0, meaning no limit. This parameter is static. Database instance needs to be restarted to make this parameter effective.

DB2_HADR_ROS

- Operating system: All
- Default: OFF Values: OFF or ON
- This variable enables the HADR reads on standby feature. When **DB2_HADR_ROS** is enabled on the HADR standby database, the standby accepts client connections and allows read-only queries to run on it.

DB2_HADR_ROS is a static registry variable, so it requires the DB2 instance to be restarted for a changed setting to take effect.

DB2_HADR_SORCVBUF

- Operating system: All
- Default: Operating system TCP socket receive buffer size, Values: `1024` to `4294967295`
- This variable specifies the operating system (OS) TCP socket receive buffer size for the HADR connection, which allows users to customize the HADR TCP/IP behavior distinctly from other connections. Some operating systems will automatically round or silently cap the user specified value. The actual buffer size used for the HADR connection is logged in the **db2diag** log files. Consult your operating system network tuning guide for the optimal setting for this parameter based on your network traffic. This variable should be used in conjunction with **DB2_HADR_SOSNDBUF**.

DB2_HADR_SOSNDBUF

- Operating system: All
- Default: Operating system TCP socket send buffer size, Values: `1024` to `4294967295`
- This variable specifies the operating system (OS) TCP socket send buffer size for the HADR connection, which allows users to customize the HADR TCP/IP behavior distinctly from other connections. Some operating systems will automatically round or silently cap the user specified value. The actual buffer size used for the HADR connection is logged in the **db2diag** log files. Consult your operating system network tuning guide for the optimal setting for this parameter based on your network traffic. This variable should be used in conjunction with **DB2_HADR_SORCVBUF**.

DB2_HISTORY_FILTER

- Operating system: All
- Default: `NULL`, Values: NULL, `G`, `L`, `Q`, `T`, `U`
- This variable specifies operations that are not to modify the history file. You can use the **DB2_HISTORY_FILTER** registry variable to reduce potential contention on the history file by filtering out operations. Specify which operations that cannot modify the history file using a comma separated list:

  ```
  db2set DB2_HISTORY_FILTER=T, L
  ```

  Possible values for **DB2_HISTORY_FILTER** are:
  - **G**: Reorg operations
  - **L**: Load operations
  - **Q**: Quiesce operations
  - **T**: Alter table space operations
  - **U**: Unload operations

DB2_INDEX_PCTFREE_DEFAULT

- Operating system: All
- Default: Not set, Values: `0` to `99`
- This registry variable specifies what percentage of each index page to leave as free space when building the index. The setting for **DB2_INDEX_PCTFREE_DEFAULT** is overridden if you explicitly specify the

PCTFREE clause on the CREATE INDEX statement. The registry variable does not affect the LEVEL2 PCTFREE clause on the CREATE INDEX statement.

The registry variable does not apply at database upgrade time, even if the indexes are re-created during the upgrade. It only applies to a new installation or once the upgrade is complete. This registry variable is dynamic; you can set it or unset it without having to stop and start instance.

If **DB2_WORKLOAD** is set to SAP, **DB2_INDEX_PCTFREE_DEFAULT** will be set to 0.

DB2LDAP_BASEDN

- Operating system: All
- Default: NULL, Values: Any valid base distinguished domain name.
- When this is set, the LDAP objects for DB2 will be stored in the LDAP directory under

```
CN=System
CN=IBM
CN=DB2
```

  under the base distinguished name specified.

  When this is set for the Microsoft Active Directory Server, ensure that CN=DB2, CN=IBM, and CN=System are defined under this distinguished name.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2LDAPCACHE

- Operating system: All
- Default: YES, Values: YES or NO
- Specifies that the LDAP cache is to be enabled. This cache is used to catalog the database, node, and DCS directories on the local machine.

  To ensure that you have the latest entries in the cache, do the following:

```
REFRESH LDAP IMMEDIATE ALL
```

  This command updates and removes incorrect entries from the database directory and the node directory.

DB2LDAP_CLIENT_PROVIDER

- Operating system: Windows
- Default: NULL (Microsoft, if available, is used; otherwise IBM is used.) Values: IBM or Microsoft
- When running in a Windows environment, DB2 supports using either Microsoft LDAP clients or IBM LDAP clients to access the LDAP directory. This registry variable is used to explicitly select the LDAP client to be used by DB2.

  Note: To display the current value of this registry variable, use the **db2set** command:

```
db2set DB2LDAP_CLIENT_PROVIDER
```

DB2LDAPHOST

- Operating system: All

- Default:Null, Values: *base_domain_name[:port_number]*, or *base_domain_name*:SSL:636 when using an SSL enabled LDAP host
- Specifies the host name and optional port number of the location for the LDAP directory where *base_domain_name* is the TCP/IP host name, and *[:port_number]* is the port number.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2LDAP_KEEP_CONNECTION

- Operating system: All
- Default: YES, Values: YES or NO
- Specifies whether DB2 caches its internal LDAP connection handles. When this variable is set to NO, DB2 will not cache its LDAP connection handles to the directory server. This will likely result in a negative performance impact, but it might be desirable to set **DB2LDAP_KEEP_CONNECTION** to NO if the number of simultaneously active LDAP client connections to the directory server needs to be minimized.

  To maximize performance, this variable is set to YES by default.

  The **DB2LDAP_KEEP_CONNECTION** registry variable is only implemented as a global level profile registry variable in LDAP, so you must set it by specifying the **-gl** option with the **db2set** command as follows:

  ```
  db2set -gl DB2LDAP_KEEP_CONNECTION=NO
  ```

DB2LDAP_SEARCH_SCOPE

- Operating system: All
- Default: DOMAIN, Values: LOCAL, DOMAIN, or GLOBAL
- Specifies the search scope for information found in database partitions or domains in the Lightweight Directory Access Protocol (LDAP). LOCAL disables searching in the LDAP directory. DOMAIN only searches in LDAP for the current directory partition. GLOBAL searches in LDAP in all directory partitions until the object is found.

DB2_LIMIT_FENCED_GROUP

- Operating system: Windows
- Default: NULL, Values: ON or OFF
- If you have Extended Security enabled, you can restrict the operating system's privileges of the fenced mode process (**db2fmp**) to the privileges assigned to the DB2USERS group by setting this registry variable to ON and by adding the DB2 service account (the user name that runs the DB2 service) to the DB2USERS group.

  Note: If LocalSystem is being used as the DB2 service account, setting **DB2_LIMIT_FENCED_GROUP** will have no effect.

  You can grant additional operating system privileges to the **db2fmp** process by adding the DB2 service account to an operating system group that holds those additional privileges.

DB2_LOAD_COPY_NO_OVERRIDE

- Operating system: All
- Default: NONRECOVERABLE, Values: COPY YES or NONRECOVERABLE
- This variable will convert any **LOAD COPY NO** to either **LOAD COPY YES** or **NONRECOVERABLE**, depending on the value of the variable. This variable is applicable to HADR primary databases as well as to standard

(non-HADR) databases; it is ignored on an HADR standby database. On an HADR primary database, if this variable is not set, **LOAD COPY NO** is converted to **LOAD NONRECOVERABLE**. The value of this variable either specifies a nonrecoverable load or the copy destination, using the same syntax as a **COPY YES** clause.

- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2LOADREC

- Operating system: All
- Default: NULL
- Used to override the location of the load copy during roll forward. If the user has changed the physical location of the load copy, **DB2LOADREC** must be set before issuing the roll forward.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2LOCK_TO_RB

- Operating system: All
- Default: NULL, Values: STATEMENT
- Specifies whether lock timeouts cause the entire transaction to be rolled back, or only the current statement. If **DB2LOCK_TO_RB** is set to STATEMENT, locked timeouts cause only the current statement to be rolled back. Any other setting results in transaction rollback.

DB2_MAP_XML_AS_CLOB_FOR_DLC

- Operating system: All
- Default: NO, Values: YES or NO
- The **DB2_MAP_XML_AS_CLOB_FOR_DLC** registry variable provides the ability to override the default DESCRIBE and FETCH behavior of XML values for clients (or DRDA Application Requestors) that do not support XML as a data type. The default value is NO, which specifies that for these clients a DESCRIBE of XML values will return BLOB(2GB), and a FETCH of XML values will result in an implicit XML serialization to BLOB that includes an XML declaration indicating an encoding of UTF-8.

  When the value is YES, a DESCRIBE of XML values will return CLOB(2GB), and a FETCH of XML values will result in an implicit XML serialization to CLOB that does not contain an XML declaration.

Note: DB2_MAP_XML_AS_CLOB_FOR_DLC is deprecated and will be removed in a future release. This variable is no longer necessary because most existing DB2 applications that access XML values do so with an XML capable client.

DB2_MAX_LOB_BLOCK_SIZE

- Operating system: All
- Default: 0 (no limit), Values: 0 to 21487483647
- Sets the maximum amount of LOB or XML data to be returned in a block. This is not a hard maximum; if this maximum is reached on the

server during data retrieval, the server finishes writing out the current row before generating a reply for the command, such as FETCH, to the client.

DB2_MEMORY_PROTECT

- Operating system: AIX with storage key support
- Default: NO, Values: NO or YES
- This registry variable enables a memory protection feature that uses storage keys to prevent data corruption in the buffer pool caused by invalid memory 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. When **DB2_MEMORY_PROTECT** is set to YES, any time a DB2 engine thread tries to illegally access buffer pool memory, that engine thread traps.

  Note: You will not be able to use the memory protection if **DB2_LGPAGE_BP** is set to YES. Even if **DB2_MEMORY_PROTECT** is set to YES, DB2 will fail to protect the buffer pool memory and disable the feature.

DB2_MIN_IDLE_RESOURCES

- Operating system: Linux
- Default: OFF, Values: OFF or ON
- This variable specifies that an activated database is to use minimal processing resources when it is idle. This might be useful in some virtual Linux environments (for example, zVM) where the small resource savings can help the host virtual machine monitor schedule its CPU and memory resources across all its virtual machines more efficiently.

DB2_NCHAR_SUPPORT

- Operating system: All
- Default: ON, Values: ON or OFF
- When this variable is set to ON (the default), the NCHAR, NVARCHAR and NCLOB spellings for the graphic data types are available for use in Unicode databases. Various national character related functions such as NCHAR() and TO_NCHAR() are also available.

  This variable should only be set to OFF if an existing database has user defined types named NCHAR, NVARCHAR, or NCLOB.

  Note: The **DB2_NCHAR_SUPPORT** registry variable may be removed in a future release, at which point you will not be able to have any user defined types named NCHAR, NVARCHAR or NCLOB in the database.

DB2NOEXITLIST

- Operating system: All
- Default: OFF, Values: ON or OFF
- This variable indicates that DB2 should not load an exit list handler and that it should not perform a commit when the application exits, regardless of the setting of the **DB2_COMMIT_ON_EXIT** registry variable.

  When **DB2NOEXITLIST** is turned off and **DB2_COMMIT_ON_EXIT** is turned on, any in-flight transactions for embedded SQL applications are automatically committed. It is recommended to explicitly add COMMIT or ROLLBACK statements when an application exits.

  Applications that dynamically load and unload the DB2 library before the application terminates might crash when calling the DB2 exit

handler. This crash might happen because the application attempts to call a function that does not exist in memory. To avoid this situation, set the **DB2NOEXITLIST** registry variable.

DB2_NUM_CKPW_DAEMONS

- Operating system: Linux and UNIX
- Default: 3, Values: 1[:FORK] to 100[:FORK]
- You can use the **DB2_NUM_CKPW_DAEMONS** registry variable to start a configurable number of check password daemons. The daemons are created during **db2start** and handle check password requests when the default **IBMOSauthserver** security plugin is in use. Increasing the setting for **DB2_NUM_CKPW_DAEMONS** can decrease the time required to establish a database connection, but this is only effective in scenarios where many connections are being made simultaneously and where authentication is expensive.

  DB2_NUM_CKPW_DAEMONS can be set to a value between 1 and 100. The database manager will create the number of daemons specified by **DB2_NUM_CKPW_DAEMONS**. Each daemon can handle check password requests directly.

  An optional FORK parameter can be added to enable the check password daemons to explicitly spawn an external check password program (**db2ckpw**) to handle check password requests. This is similar to setting **DB2_NUM_CKPW_DAEMONS** to zero in previous releases. In FORK mode, each check password daemon will spawn the check password program for each request to check a password. The daemons in FORK mode are started as the instance owner.

  If **DB2_NUM_CKPW_DAEMONS** is set to zero, the effective value is set to 3:FORK, where 3 check password daemons are started in FORK mode.

DB2_OPTSTATS_LOG

- Operating system: All
- Default: Not set (see details below), Values: OFF, ON {NUM | SIZE | NAME | DIR}
- **DB2_OPTSTATS_LOG** specifies the attributes of the statistics event logging files which are used to monitor and analyze statistics collection related activities. When **DB2_OPTSTATS_LOG** is not set or set to ON, statistics event logging is enabled, allowing you to monitor system performance and keep a history for better problem determination. Log records are written to the first log file until it is full. Subsequent records are written to the next available log file. If the maximum number of files is reached, the oldest log file will be overwritten with the new records. If system resource consumption is of great concern to you, disable this registry variable by setting it to OFF.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

When statistics event logging is explicitly enabled (set to ON), there are a number of options you can modify:

- NUM: the maximum number of rotating log files. Default: 5, Values 1 - 15
- SIZE: the maximum size of rotating log files. (The size of each rotating file is SIZE/NUM.) Default = 15 Mb, Values 1 Mb – 4096 Mb

- NAME: the base name for rotating log files. Default: db2optstats.*number*.log, for instance db2optstats.0.log, db2optstats.1.log, etc.
- DIR: the base directory for rotating log files. Default: **diagpath**/events

You can specify a value for as many of these options as you want, just ensure that ON is the first value when you want to enable statistics logging. For instance, to enable statistics logging with maximum of 6 log files, a maximum file size of 25 Mb, a base file name of mystatslog, and the directory mystats, issue the following command:

```
db2set DB2_OPTSTATS_LOG=ON,NUM=6,SIZE=25,NAME=mystatslog,DIR=mystats
```

DB2REMOTEPREG

- Operating system: Windows
- Default: NULL, Values: Any valid Windows computer name
- Specifies the remote computer name that contains the Win32 registry list of DB2 instance profiles and DB2 instances. The value for **DB2REMOTEPREG** must be set only once after the DB2 database product is installed, and must not be changed after it is set. Use this variable with extreme caution.
- In a partitioned database environment, you can 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. See "Setting variables at the instance level in a partitioned database environment" on page 546 for more information about when to use this variable.

  When the DB2 database manager reads the registry variables on Windows operating systems, it reads the **DB2REMOTEPREG** value first. If the **DB2REMOTEPREG** variable is set, the database manager opens the registry on the remote computer that is specified by the **DB2REMOTEPREG** variable. Subsequent reading and updating of the registry variables is redirected to the specified remote computer.

  For a computer that is not the instance owner to access the remote registry, the Remote Registry Service must be running on the target computer. Also, the user logon account and all DB2 service logon accounts must have sufficient access to the remote registry. To use the **DB2REMOTEPREG** variable, you must operate in a Windows domain environment so that you can grant the required registry access to the domain account.
- Do not use **DB2REMOTEPREG** in a Microsoft Cluster Server environment.

DB2_RESOLVE_CALL_CONFLICT

- Operating system: AIX, HP-UX, Solaris, Linux, Windows
- Default: YES, Values: YES, NO
- When routines called by triggers attempt to access tables that have been modified by other statements or routines in the body of the trigger, this can break nested SQL statement rules. Setting **DB2_RESOLVE_CALL_CONFLICT** enforces that all modifications to tables are completed in compliance with the SQL standard rules for triggers before executing the CALL statement.

  You must stop the instance before you reset **DB2_RESOLVE_CALL_CONFLICT** and then restart it. Then rebind any packages which cause invocation of triggers. To rebind SQL Procedures use: CALL

SYSPROC.REBIND_ROUTINE_PACKAGE
('P','*procedureschema.procedurename*','CONSERVATIVE');

You need to be aware that **DB2_RESOLVE_CALL_CONFLICT** can have a performance impact. Setting **DB2_RESOLVE_CALL_CONFLICT** to YES causes the DB2 database manager to resolve all potential read and write conflicts through the injection of temporary tables, as needed. Typically, the impact is small because at most one temporary table is injected. This has a small effect in an OLTP environment because only one (or a small number of) rows are being modified by the triggering statement. Typically, when following the general recommendation to use SMS (system managed space) for temporary table spaces, the performance impact from setting **DB2_RESOLVE_CALL_CONFLICT** is expected to be low.

Changes to this variable can take effect immediately for all future compiled SQL statements if the **db2set** command is issued with the **-immediate** parameter. There is no need to restart the instance.

DB2_RESTRICT_DDF

- Operating system: All
- Default: FALSE, Values: TRUE or FALSE
- Specifies whether the dynamic data format feature, also known as *progressive streaming* should be disabled. When **DB2_RESTRICT_DDF** is set to TRUE, the server informs the JDBC driver that the dynamic data format feature is to be disabled.

  In SAP environments, when **DB2_WORKLOAD**=SAP is set, the default value of this registry variable is TRUE.

DB2ROUTINE_DEBUG

- Operating system: AIX and Windows
- Default: OFF, Values: ON or OFF
- Specifies whether to enable the debug capability for Java stored procedures. If you are not debugging Java stored procedures, use the default, OFF. There is a performance impact to enable debugging.

Note: DB2ROUTINE_DEBUG is deprecated and will be removed in a future release. This stored procedure debugger has been replaced by the Unified Debugger.

DB2_SAS_SETTINGS

- Operating system: All
- Default: Not set. Values: ENABLE_SAS_EP, LIBRARY_PATH, COMM_BUFFER_SZ, COMM_TIMEOUT, RESTART_RETRIES, DIAGPATH, DIAGLEVEL
- This variable is the primary point of configuration for in-database analytics with the SAS embedded process (EP). All options except for the ENABLE_SAS_EP option are configurable online.

  ENABLE_SAS_EP
  If you set this option to TRUE, the SAS EP starts automatically when you issue the **db2start** command. The default for this option is FALSE.

  LIBRARY_PATH
  The fully qualified path from which to load the SAS EP library the next time that the SAS EP process starts. If you do not specify a path, the DB2 database manager looks for the SAS EP library under the sqllib/function/sas directory. For security reasons, you should

install the SAS EP library in a location where unauthorized users cannot modify or replace the file. Choose one of the following options:

- Ensure that the library path and the SAS EP library file are owned and can be written to only by the instance owner.
- Place the file in a directory, such as `sqllib/function`, that has the sticky bit set.

Only a user with SYSADM authority can configure the library path by using the **db2set** command.

COMM_BUFFER_SZ
An integer value specifying the amount of shared memory buffer, in 4 KB pages, to use for communication sessions between the DB2 data server and the SAS EP. The valid range of values for this parameter is 1 - 32767. The default value is 15. Communications buffers are allocated from the FMP communications heap. For more information, see **DB2_FMP_COMM_HEAPSZ**.

COMM_TIMEOUT
A timeout value that the DB2 database manager uses to determine whether the SAS EP should be considered unresponsive when exchanging control messages. If this value is reached, the database manager kills the SAS EP so that it can be spawned again. The default is 300 seconds.

RESTART_RETRIES
An integer value specifying the number of times that the DB2 database manager attempts to respawn the SAS EP after detecting that it has terminated abnormally. The valid range of values for this parameter is 0 - 100. The default value is 10. After the retry count has been reached, the database manager waits for 15 minutes before attempting the operation again.

DIAGPATH
A fully qualified path specifying the location of diagnostic logs for the SAS EP. The default value is the value of the **diagpath** database manager configuration parameter.

DIAGLEVEL
An integer value specifying the minimum severity level of messages that are captured in the SAS diagnostic logs. The valid values for this option are as follows:

1 Severe

2 Error

3 Warning

4 Informational

The default value is the value of the **diaglevel** database manager configuration parameter.

MEMSIZE
An integer value specifying the maximum amount of memory, in 4 KB pages, that the SAS EP can consume on a particular host. The valid range of values for this option is 1 - 4294967295. If there are multiple logical partitions, the value that is applied to each partition is divided by the number of logical partitions on the corresponding

host. The default value is 20% of the value of the **instance_memory** database manager configuration parameter. If you set the **instance_memory** parameter to a fixed value, ensure that this value takes the additional memory requirements for the SAS EP into account.

Example:

```
db2set DB2_SAS_SETTINGS="ENABLE_SAS_EP:TRUE;
  LIBRARY_PATH:/home/instowner/sqllib/function/sas"
```

DB2SATELLITEID

- Operating system: All
- Default: NULL, Values: a valid satellite ID declared in the Satellite Control Database
- Specifies the satellite ID that is passed to the satellite control server when a satellite synchronizes. If a value is not specified for this variable, the logon ID is used as the satellite ID.

DB2_SERVER_CONTIMEOUT

- Operating system: All
- Default: 180, Values: 0 to 32767 seconds
- This registry variable and the **DB2_DISPATCHER_PEEKTIMEOUT** registry variable both configure the handling of a new client during connect time. **DB2_SERVER_CONTIMEOUT** allows you to adjust the time, in seconds, that an agent waits for a client's connection request before terminating the connection. In most cases, you should not need to adjust this registry variable, but if DB2 clients are constantly being timed out by the server at connect time, you can set a higher value for **DB2_SERVER_CONTIMEOUT** to extend the timeout period. If an invalid value is set, the default value is used. This registry variable is not dynamic.

DB2_SERVER_ENCALG

- Operating system: All
- Default: NULL, Values: AES_CMP or AES_ONLY
-

  Note: DB2_SERVER_ENCALG is deprecated in Version 9.7 and might be removed in a future release.

  If the **DB2_SERVER_ENCALG** registry variable is set when you upgrade your instances to DB2 Version 9.7, the **alternate_auth_enc** configuration parameter is set to AES_ONLY or AES_CMP according to the setting of **DB2_SERVER_ENCALG**. Thereafter, to specify the encryption algorithm for encrypting user IDs and passwords, update the **alternate_auth_enc** configuration parameter. If the **alternate_auth_enc** configuration parameter is set, its value takes precedence over the **DB2_SERVER_ENCALG** registry variable value.

DB2SORT

- Operating system: All, server only
- Default: NULL
- This variable specifies the location of a library to be loaded at runtime by the load utility. The library contains the entry point for functions used in sorting indexing data. Use **DB2SORT** to exploit vendor-supplied sorting products for use with the load utility in generating table indexes. The path supplied must be relative to the database server.

DB2_STANDBY_ISO

- Operating system: All
- Default: NULL, Values: UR or OFF
- This variable coerces the isolation level requested by applications and statements running on an active HADR standby database to Uncommitted Read (UR). When **DB2_STANDBY_ISO** is set to UR, isolation levels higher than UR are coerced to UR with no warning returned. If the HADR standby takes over as the HADR primary, this variable will have no effect.

DB2STMM

- Operating system: UNIX
- The registry variable controls a set of parameters which allow you to modify certain characteristics of the self tuning memory manager (STMM).
- Parameters:

  GLOBAL_BENEFIT_SEGMENT_COMPATIBLE

  – Default: Not set, Values: YES, NO

  – The GLOBAL_BENEFIT_SEGMENT_COMPATIBLE parameter only has a functional impact if the **database_memory** configuration parameter is set to AUTOMATIC for a database.

  This parameter influences the permission settings of the STMM shared memory segment. It should only be set to YES on systems with multiple instances, where some of the instances are downlevel and have **database_memory** set to AUTOMATIC, in order to mitigate downlevel compatibility issues that impact the tuning of a database's overall database memory usage. A downlevel instance would be an instance belonging to any of the following DB2 releases and fix pack levels: DB2 V9.1 at all fix pack levels, DB2 V9.5 fix pack 7 and earlier, and DB2 V9.7 fix pack 4 and earlier.

  For instances that are non-root DB2 installations, you should set this variable only if you want all instances on the system make use of the same STMM shared memory segment. Leaving this variable unset or set to NO will cause a non-root instance to use its own instance-specific STMM shared memory segment, which may impact the tuning of overall database memory usage for any databases with **database_memory** set to AUTOMATIC.

  This registry variable is read once, during the DB2 instance startup. Note that you need to set this parameter across all the upgraded (that is, non-downlevel) instances and once set, you need to restart all upgraded instances.

  GLOBAL_BENEFIT_SEGMENT_UNIQUE

  – Default: Not set, Values: YES, NO

  – The GLOBAL_BENEFIT_SEGMENT_UNIQUE parameter only has a functional impact if the **database_memory** configuration parameter is set to AUTOMATIC for a database.

  This parameter specifies that each upgraded (that is, non-downlevel) instance is to make use of its own instance-specific STMM shared memory segment. The means that each instance tunes overall database memory usage for any of the

databases belonging to it, independent of the tuning of overall database memory usage of databases belonging to the other instances on the system.

You should only consider setting this parameter to YES if the **instance_memory** configuration parameter is **not** set to AUTOMATIC for all instances on a system.

This registry variable is read once, during the DB2 instance startup. Note that this parameter needs to be set across all the upgraded instances and, once set, it requires that you restart all upgraded instances.

DB2_TRUNCATE_REUSESTORAGE

- Operating system: All
- Default: NULL (not set), Values: IMPORT, import
- You can use this variable to resolve lock contention between the **IMPORT** with **REPLACE** command and the **BACKUP ... ONLINE** command. In some situations, online backup and truncate operations are unable to execute concurrently. When this occurs, you can set **DB2_TRUNCATE_REUSESTORAGE** to IMPORT or import, and physical truncation of the object, including data, indexes, long fields, large objects and block maps (for multidimensional clustering tables), is skipped and only logical truncation is performed. That is, the **IMPORT** with **REPLACE** command empties the table, causing the object's logical size to decrease, but the storage on disk remains allocated.

  This registry variable is dynamic; you can set it or unset it without having to stop and start instance. You can set **DB2_TRUNCATE_REUSESTORAGE** before an online backup starts and then unset it after online backup completes. For multi-partitioned environments, the registry variable will only be active on the nodes on which the variable is set. **DB2_TRUNCATE_REUSESTORAGE** is only effective on DMS permanent objects.

  In SAP environments, when **DB2_WORKLOAD**=SAP is set, the default value of this registry variable is IMPORT.
- Changes to this variable will take effect immediately for all future compiled SQL statements. There is no need to restart the instance or to issue the **db2set** command with the **-immediate** parameter.

DB2_UTIL_MSGPATH

- Operating system: All
- Default: *instanceName*/tmp directory
- The **DB2_UTIL_MSGPATH** registry variable is used in conjunction with the SYSPROC.ADMIN_CMD procedure, the SYSPROC.ADMIN_REMOVE_MSGS procedure, and the SYSPROC.ADMIN_GET_MSGS UDF. It applies on the instance level. **DB2_UTIL_MSGPATH** can be set to indicate a directory path on the server where the fenced user ID can read, write and delete files. This directory must be accessible from all coordinator partitions, and must have sufficient space to contain utility message files.

  If this path is not set, the *instanceName*/tmp directory is used as the default (note that *instanceName*/tmp is cleaned up when DB2 is uninstalled).

  If this path is not set when the ALTOBJ procedure is run, a temporary message file is created in the ~sqllib/tmp directory.

If this path is changed, the files that existed in the directory pointed to by the previous setting are not automatically moved or deleted. If you want to retrieve the contents of the messages created under the old path, you must manually move these messages (which are prefixed with the utility name and suffixed with the user ID) to the new directory to which **DB2_UTIL_MSGPATH** points. The next utility message file is created, read, and cleaned up in the new location.

The files under the **DB2_UTIL_MSGPATH** directory are utility specific, not transaction dependent. They are not part of the backup image. The files under the **DB2_UTIL_MSGPATH** directory are user managed; that means a user can delete the message files using the SYSPROC.ADMIN_REMOVE_UTILMSG procedure. These files are not cleaned up by uninstalling DB2.

DB2_XBSA_LIBRARY

- Operating system: AIX, HP-UX, Solaris, and Windows
- Default: NULL, Values: Any valid path and file.
- Points to the vendor-supplied XBSA library. On AIX, the setting must include the shared object if it is not named shr.o. HP-UX, Solaris, and Windows do not require the shared object name. For example, to use Legato's NetWorker Business Suite Module for DB2, the registry variable must be set as follows:

  ```
  db2set DB2_XSBA_LIBRARY="/usr/lib/libxdb2.a(bsashr10.o)"
  ```

  The XBSA interface can be invoked through the **BACKUP DATABASE** or the **RESTORE DATABASE** commands. For example:

  ```
  db2 backup db sample use XBSA
  db2 restore db sample use XBSA
  ```

DB2_XSLT_ALLOWED_PATH

- Operating system: All
- Default: NULL or NONE, Values: ALL or a list of valid URIs , separated by a whitespace
- This registry variable controls how the DB2 instance refers to the external entities defined inside of an XSLT stylesheet.
  - NULL or NONE: No URI references are permitted, and the transformation with such a stylesheet fails.
  - ALL: All references to URIs are allowed.

    Note: Uncontrolled reference to an external URI might be a severe security issue.
  - List of URIs: Only references to URIs that are located in subdirectories of the URIs from the list are allowed, as shown in the following example:

    ```
    db2set DB2_XSLT_ALLOWED_PATH =" http://some.website.com/test/dir   /home/Joe/resource
    ```

Chapter 24. Configuration parameters

When a DB2 database instance or a database is created, a corresponding 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; to 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 runtime 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`. These files cannot be directly edited, and can only be changed or viewed 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 will be 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 may be required, depending on the type of activity in a specific database.

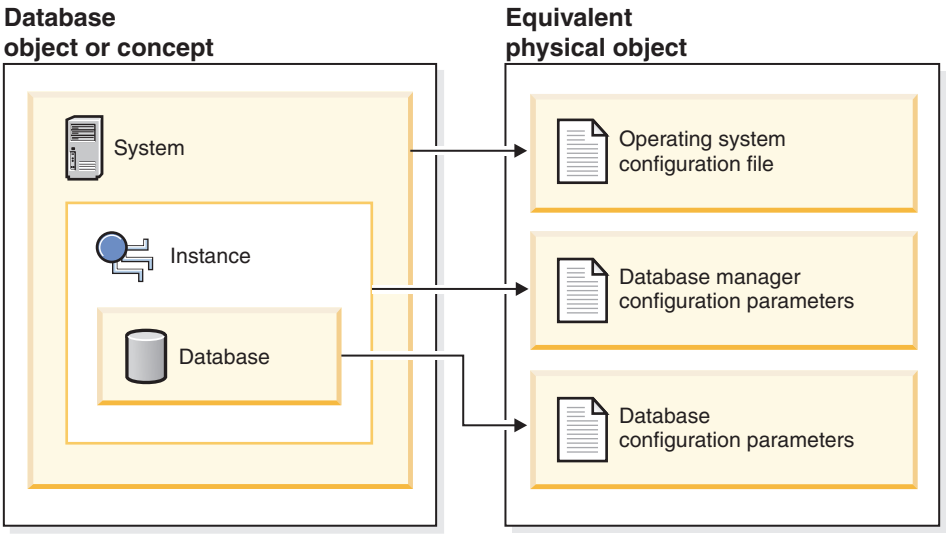


Figure 52. Relationship between database objects and configuration files

Configuring the DB2 database manager with 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`.

Configuration parameters summary

The following tables list the parameters in the database manager and database configuration files for database servers. When changing the database manager and database configuration parameters, consider the detailed information for each parameter. Specific operating environment information including defaults is part of each parameter description.

Database Manager Configuration Parameter Summary

For some database manager configuration parameters, the database manager must be stopped (**db2stop**) and restarted (**db2start**) for the new parameter values to take effect. Other parameters can be changed online; these are called *configurable online configuration parameters*. 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 applies the change immediately. If you do not want the change applied immediately, use the **DEFERRED** option on the **UPDATE DBM CFG** command.

The column "Auto" in the following table indicates whether the parameter supports the **AUTOMATIC** keyword on the **UPDATE DBM CFG** command.

When updating a parameter to automatic, it is also possible to specify a starting value as well as the **AUTOMATIC** keyword. Note that the value can mean something different for each parameter, and in some cases it is not applicable. Before specifying a value, read the parameter's documentation to determine what it represents. In the following example, **num_poolagents** will be updated to AUTOMATIC and the database manager will use 20 as the minimum number of idle agents to pool:

```
db2 update dbm cfg using num_poolagents 20 automatic
```

To unset the AUTOMATIC feature, the parameter can be updated to a value or the **MANUAL** keyword can be used. When a parameter is updated to MANUAL, the parameter is no longer automatic and is set to its current value (as displayed in the Current Value column from the **GET DBM CFG SHOW DETAIL** and **GET DB CFG SHOW DETAIL** commands).

If a database is created by either the **CREATE DATABASE**, or the sqlecrea API, then the Configuration Advisor runs by default to update the database configuration parameters with automatically computed values. If a database is created by either the **CREATE DATABASE** command with the AUTOCONFIGURE APPLY NONE clause added, or the sqlecrea API specifies not to run the Configuration Advisor, then the configuration parameters are set to the default values.

The column "Perf. Impact" provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High** — Indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters, which, in some cases, means that you will accept the default values provided.
- **Medium** — Indicates that the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- **Low** — Indicates that the parameter has a less general or less significant impact on performance.
- **None** — Indicates that the parameter does not directly impact performance. Although you do not have to tune these parameters for performance enhancement, they can be very important for other aspects of your system configuration, such as communication support, for example.

The columns "Token", "Token Value", and "Data Type" provide information that you will need when calling the db2CfgGet or the db2CfgSet API. This information includes configuration parameter identifiers, entries for the *token* element in the db2CfgParam data structure, and data types for values that are passed to the structure.

Table 124. Configurable Database Manager Configuration Parameters

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
agent_stack_sz	No	No	Low	SQLF_KTN_AGENT_STACK_SZ	61	Uint16	"agent_stack_sz - Agent stack size" on page 661
agentpri	No	No	High	SQLF_KTN_AGENTPRI	26	Sint16	"agentpri - Priority of agents" on page 663
alt_diagpath	Yes	No	None	SQLF_KTN_ALT_DIAGPATH SQLF_KTN_ALT_DIAGPATH_FULL	941	char [] (String)	"alt_diagpath - Alternate diagnostic data directory path" on page 664
alternate_auth_enc [6]	No	No	Low	SQLF_KTN_ALTERNATE_AUTH_ENC	938	Uint16	"alternate_auth_enc - Alternate encryption algorithm for incoming connections at server configuration parameter" on page 666
aslheapsz	No	No	High	SQLF_KTN_ASLHEAPSZ	15	Uint32	"aslheapsz - Application support layer heap size" on page 667

Table 124. Configurable Database Manager Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
audit_buf_sz	No	No	High	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32	"audit_buf_sz - Audit buffer size" on page 668
authentication	No	No	Low	SQLF_KTN_ AUTHENTICATION	78	Uint16	"authentication - Authentication type" on page 669
catalog_noauth	Yes	No	None	SQLF_KTN_CATALOG_ NOAUTH	314	Uint16	"catalog_noauth - Cataloging allowed without authority" on page 674
cf_diaglevel	No	No	None	SQLF_KTN_CF_DIAGLEVEL	968	Uint16	"cf_diaglevel - diagnostic error capture level configuration parameter for the CF" on page 670
cf_diagpath	No	No	None	SQLF_KTN_CF_DIAGPATH SQLF_KTN_CF_DIAGPATH_ FULL	969	char(215)	"cf_diagpath - diagnostic data directory path configuration parameter for the CF" on page 671
cf_mem_sz	No	Yes	High	SQLF_KTN_CF_MEM_SZ	960	Uint32	"cf_mem_sz - CF memory configuration parameter" on page 672
cf_num_conns	Yes	Yes	High	SQLF_KTN_CF_NUM_CONNS	966	Uint32	"cf_num_conns - Number of CF connections per member per CF configuration parameter" on page 672
cf_num_workers	No	Yes	High	SQLF_KTN_CF_NUM_WORKERS	961	Uint32	"cf_num_workers - Number of worker threads configuration parameter" on page 673
clnt_krb_plugin	No	No	None	SQLF_KTN_CLNT_KRB_ PLUGIN	812	char(33)	"clnt_krb_plugin - Client Kerberos plug-in" on page 675
clnt_pw_plugin	No	No	None	SQLF_KTN_CLNT_PW_ PLUGIN	811	char(33)	"clnt_pw_plugin - Client userid-password plug-in" on page 675
cluster_mgr	No	No	None	SQLF_KTN_CLUSTER_MGR	920	char(262)	"cluster_mgr - Cluster manager name" on page 676
comm_bandwidth	Yes	No	Medium	SQLF_KTN_COMM_ BANDWIDTH	307	float	"comm_bandwidth - Communications bandwidth" on page 676
comm_exit_list	No	No	Low	SQLF_KTN_COMM_EXIT_LIST	10121	char(129)	"comm_exit_list - Communication buffer exit library list" on page 677
conn_elapse	Yes	No	Medium	SQLF_KTN_CONN_ELAPSE	508	Uint16	"conn_elapse - Connection elapse time" on page 677
cpuspeed	Yes	No	High	SQLF_KTN_CPUSPEED	42	float	"cpuspeed - CPU speed" on page 678
dft_account_str	Yes	No	None	SQLF_KTN_DFT_ ACCOUNT_STR	28	char(25)	"dft_account_str - Default charge-back account" on page 679
dft_monswitches • **dft_mon_bufpool** • **dft_mon_lock** • **dft_mon_sort** • **dft_mon_stmt** • **dft_mon_table** • **dft_mon_ timestamp** • **dft_mon_uow**	Yes	No	Medium	SQLF_KTN_DFT_ MONSWITCHES[2] • SQLF_KTN_DFT_MON_ BUFPOOL • SQLF_KTN_DFT_MON_LOCK • SQLF_KTN_DFT_MON_SORT • SQLF_KTN_DFT_MON_STMT • SQLF_KTN_DFT_MON_ TABLE • SQLF_KTN_DFT_MON_ TIMESTAMP • SQLF_KTN_DFT_MON_ UOW	29 • 33 • 34 • 35 • 31 • 32 • 36 • 30	Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16	"dft_monswitches - Default database system monitor switches" on page 680
dftdbpath	Yes	No	None	SQLF_KTN_DFTDBPATH	27	char(215)	"dftdbpath - Default database path" on page 681
diaglevel	Yes	No	Low	SQLF_KTN_DIAGLEVEL	64	Uint16	"diaglevel - Diagnostic error capture level" on page 682
diagpath	Yes	No	None	SQLF_KTN_DIAGPATH SQLF_KTN_DIAGPATH_FULL	65	char(215)	"diagpath - Diagnostic data directory path" on page 683
dir_cache	No	No	Medium	SQLF_KTN_DIR_CACHE	40	Uint16	"dir_cache - Directory cache support" on page 688
discover[3]	No	No	Medium	SQLF_KTN_DISCOVER	304	Uint16	"discover - Discovery mode" on page 690
discover_inst	Yes	No	Low	SQLF_KTN_DISCOVER_INST	308	Uint16	"discover_inst - Discover server instance" on page 690
fcm_num_buffers	Yes	Yes	Medium	SQLF_KTN_FCM_NUM_ BUFFERS	503	Uint32	"fcm_num_buffers - Number of FCM buffers" on page 691
fcm_num_ channels	Yes	Yes	Medium	SQLF_KTN_FCM_NUM_ CHANNELS	902	Uint32	"fcm_num_channels - Number of FCM channels" on page 692
fcm_parallelism	No	No	High	SQLF_KTN_FCM_NUM_PARALLELISM	848	Sint32	"fcm_parallelism - Internode communication parallelism" on page 693
fed_noauth	Yes	No	None	SQLF_KTN_FED_NOAUTH	806	Uint16	"fed_noauth - Bypass federated authentication" on page 693
federated	Yes	No	Medium	SQLF_KTN_FEDERATED	604	Sint16	"federated - Federated database system support" on page 694
federated_async	Yes	Yes	Medium	SQLF_KTN_FEDERATED_ ASYNC	849	Sint32	"federated_async - Maximum asynchronous TQs per query configuration parameter" on page 694
fenced_pool	Yes	Yes	Medium	SQLF_KTN_FENCED_POOL	80	Sint32	"fenced_pool - Maximum number of fenced processes" on page 695
group_plugin	No	No	None	SQLF_KTN_GROUP_PLUGIN	810	char(33)	"group_plugin - Group plug-in" on page 696
health_mon	Yes	No	Low	SQLF_KTN_HEALTH_MON	804	Uint16	"health_mon - Health monitoring" on page 696
indexrec[4]	Yes	No	Medium	SQLF_KTN_INDEXREC	20	Uint16	"indexrec - Index re-creation time" on page 697

Table 124. Configurable Database Manager Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
instance_memory	Yes	Yes	Medium	SQLF_KTN_INSTANCE_ MEMORY	803	Uint64	"instance_memory - Instance memory" on page 699
intra_parallel	No	No	High	SQLF_KTN_INTRA_PARALLEL	306	Sint16	"intra_parallel - Enable intrapartition parallelism" on page 702
java_heap_sz	No	No	High	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32	"java_heap_sz - Maximum Java interpreter heap size" on page 702
jdk_path	No	No	None	SQLF_KTN_JDK_PATH	311	char(255)	"jdk_path - Software Developer's Kit for Java installation path" on page 703
keepfenced	No	No	Medium	SQLF_KTN_KEEPFENCED	81	Uint16	"keepfenced - Keep fenced process" on page 703
local_gssplugin	No	No	None	SQLF_KTN_LOCAL_ GSSPLUGIN	816	char(33)	"local_gssplugin - GSS API plug-in used for local instance level authorization" on page 705
max_connections	Yes	Yes	Medium	SQLF_KTN_MAX_ CONNECTIONS	802	Sint32	"max_connections - Maximum number of client connections" on page 705
max_connretries	Yes	No	Medium	SQLF_KTN_MAX_ CONNRETRIES	509	Uint16	"max_connretries - Node connection retries" on page 706
max_coordagents	Yes	Yes	Medium	SQLF_KTN_MAX_ COORDAGENTS	501	Sint32	"max_coordagents - Maximum number of coordinating agents" on page 706
max_querydegree	Yes	No	High	SQLF_KTN_MAX_ QUERYDEGREE	303	Sint32	"max_querydegree - Maximum query degree of parallelism" on page 708
max_time_diff	No	No	Medium	SQLF_KTN_MAX_TIME_DIFF	510	Uint16	"max_time_diff - Maximum time difference between members" on page 708
mon_heap_sz	Yes	Yes	Low	SQLF_KTN_MON_HEAP_SZ	79	Uint16	"mon_heap_sz - Database system monitor heap size" on page 711
notifylevel	Yes	No	Low	SQLF_KTN_NOTIFYLEVEL	605	Sint16	"notifylevel - Notify level" on page 712
num_initagents	No	No	Medium	SQLF_KTN_NUM_ INITAGENTS	500	Uint32	"num_initagents - Initial number of agents in pool" on page 713
num_initfenced	No	No	Medium	SQLF_KTN_NUM_ INITFENCED	601	Sint32	"num_initfenced - Initial number of fenced processes" on page 714
num_poolagents	Yes	Yes	High	SQLF_KTN_NUM_ POOLAGENTS	502	Sint32	"num_poolagents - Agent pool size" on page 714
numdb	No	No	Low	SQLF_KTN_NUMDB	6	Uint16	"numdb - Maximum number of concurrently active databases including host and System i databases" on page 716
query_heap_sz	No	No	Medium	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32	"query_heap_sz - Query heap size" on page 717
resync_interval	No	No	None	SQLF_KTN_RESYNC_ INTERVAL	68	Uint16	"resync_interval - Transaction resync interval" on page 719
rqrioblk	No	No	High	SQLF_KTN_RQRIOBLK	1	Uint16	"rqrioblk - Client I/O block size" on page 720
sheapthres	No	No	High	SQLF_KTN_SHEAPTHRES	21	Uint32	"sheapthres - Sort heap threshold" on page 720
spm_log_file_sz	No	No	Low	SQLF_KTN_SPM_LOG_FILE_ SZ	90	Sint32	"spm_log_file_sz - Sync point manager log file size" on page 722
spm_log_path	No	No	Medium	SQLF_KTN_SPM_LOG_PATH	313	char(226)	"spm_log_path - Sync point manager log file path" on page 723
spm_max_resync	No	No	Low	SQLF_KTN_SPM_MAX_ RESYNC	91	Sint32	"spm_max_resync - Sync point manager resync agent limit" on page 723
spm_name	No	No	None	SQLF_KTN_SPM_NAME	92	char(8)	"spm_name - Sync point manager name" on page 723
srvcon_auth	No	No	None	SQLF_KTN_SRVCON_AUTH	815	Uint16	"srvcon_auth - Authentication type for incoming connections at the server" on page 724
srvcon_gssplugin_ list	No	No	None	SQLF_KTN_SRVCON_ GSSPLUGIN_ LIST	814	char(256)	"srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server" on page 724
srv_plugin_mode	No	No	None	SQLF_KTN_SRV_PLUGIN_ MODE	809	Uint16	"srv_plugin_mode - Server plug-in mode" on page 725
srvcon_pw_plugin	No	No	None	SQLF_KTN_SRVCON_PW_ PLUGIN	813	char(33)	"srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server" on page 725
ssl_svr_keydb	No	No	None	SQLF_KTN_SSL_SVR_KEYDB	930	char(1023)	"ssl_svr_keydb - SSL key file path for incoming SSL connections at the server configuration parameter" on page 727
ssl_svr_stash	No	No	None	SQLF_KTN_SSL_SVR_STASH	931	char(1023)	"ssl_svr_stash - SSL stash file path for incoming SSL connections at the server configuration parameter" on page 728
ssl_svr_label	No	No	None	SQLF_KTN_SSL_SVR_LABEL	932	char(1023)	"ssl_svr_label - Label in the key file for incoming SSL connections at the server configuration parameter" on page 728
ssl_svcename	No	No	None	SQLF_KTN_SSL_SVCENAME	933	char(14)	"ssl_svcename - SSL service name configuration parameter" on page 730

Table 124. Configurable Database Manager Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
ssl_cipherspecs	No	No	None	SQLF_KTN_SSL_CIPHERSPECS	934	char(255)	"ssl_cipherspecs - Supported cipher specifications at the server configuration parameter" on page 726
ssl_versions	No	No	None	SQLF_KTN_SSL_VERSIONS	935	char(255)	"ssl_versions - Supported SSL versions at the server configuration parameter" on page 730
ssl_clnt_keydb	No	No	None	SQLF_KTN_SSL_CLNT_KEYDB	936	char(1023)	"ssl_clnt_keydb - SSL key file path for outbound SSL connections at the client configuration parameter" on page 726
ssl_clnt_stash	No	No	None	SQLF_KTN_SSL_CLNT_STASH	937	char(1023)	"ssl_clnt_stash - SSL stash file path for outbound SSL connections at the client configuration parameter" on page 727
start_stop_time	Yes	No	Low	SQLF_KTN_START_STOP_ TIME	511	Uint16	"start_stop_time - Start and stop timeout" on page 729
svcename	No	No	None	SQLF_KTN_SVCENAME	24	char(14)	"svcename - TCP/IP service name" on page 731
sysadm_group	No	No	None	SQLF_KTN_SYSADM_ GROUP	39	char(128)	"sysadm_group - System administration authority group name" on page 731
sysctrl_group	No	No	None	SQLF_KTN_SYSCTRL_ GROUP	63	char(128)	"sysctrl_group - System control authority group name" on page 732
sysmaint_group	No	No	None	SQLF_KTN_SYSMAINT_ GROUP	62	char(128)	"sysmaint_group - System maintenance authority group name" on page 733
sysmon_group	No	No	None	SQLF_KTN_SYSMON_ GROUP	808	char(128)	"sysmon_group - System monitor authority group name" on page 733
tm_database	No	No	None	SQLF_KTN_TM_DATABASE	67	char(8)	"tm_database - Transaction manager database name" on page 734
tp_mon_name	No	No	None	SQLF_KTN_TP_MON_NAME	66	char(19)	"tp_mon_name - Transaction processor monitor name" on page 734
trust_allclnts[5]	No	No	None	SQLF_KTN_TRUST_ALLCLNTS	301	Uint16	"trust_allclnts - Trust all clients" on page 736
trust_clntauth	No	No	None	SQLF_KTN_TRUST_ CLNTAUTH	302	Uint16	"trust_clntauth - Trusted clients authentication" on page 736
util_impact_lim	Yes	No	High	SQLF_KTN_UTIL_IMPACT_ LIM	807	Uint32	"util_impact_lim - Instance impact policy" on page 737
wlm_dispatcher	Yes	No	Medium	SQLF_KTN_WLM_ DISPATCHER	976	Uint16	"wlm_dispatcher - Workload management dispatcher" on page 738
wlm_disp_concur	Yes	No	Low	SQLF_KTN_WLM_DISP_ CONCUR	977	Sint16	"wlm_disp_concur - Workload manager dispatcher thread concurrency" on page 738
wlm_disp_cpu_shares	Yes	No	Low	SQLF_KTN_WLM_DISP_ CPU_SHARES	979	Uint16	"wlm_disp_cpu_shares - Workload manager dispatcher CPU shares" on page 739
wlm_disp_min_util	Yes	No	Low	SQLF_KTN_WLM_DISP_ MIN_UTIL	978	Uint16	"wlm_disp_min_util - Workload manager dispatcher minimum CPU utilization" on page 740

Note:

Please look at the header files `sqlenv.h` and `sqlutil.h` for the valid values and for the definitions used by the configuration parameters.

1.

```
Bit 1 (xxxx xxx1): dft_mon_uow
Bit 2 (xxxx xx1x): dft_mon_stmt
Bit 3 (xxxx x1xx): dft_mon_table
Bit 4 (xxxx 1xxx): dft_mon_buffpool
Bit 5 (xxx1 xxxx): dft_mon_lock
Bit 6 (xx1x xxxx): dft_mon_sort
Bit 7 (x1xx xxxx): dft_mon_timestamp
```

2. Valid values:

```
SQLF_DSCVR_KNOWN (1)
SQLF_DSCVR_SEARCH (2)
```

3. Valid values:

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
SQLF_INX_REC_RESTART (2)
SQLF_INX_REC_RESTART_NO_REDO (3)
SQLF_INX_REC_ACCESS_NO_REDO (4)
```

4. Valid values:

```
SQLF_TRUST_ALLCLNTS_NO (0)
SQLF_TRUST_ALLCLNTS_YES (1)
SQLF_TRUST_ALLCLNTS_DRDAONLY (2)
```

5. Valid values:

```
SQL_ALTERNATE_AUTH_ENC_AES (0)
SQL_ALTERNATE_AUTH_ENC_AES_CMP (1)
SQL_ALTERNATE_AUTH_ENC_NOTSPEC (255)
```

Table 125. Informational Database Manager Configuration Parameters

Parameter	Token	Token Value	Data Type	Additional Information
nodetype[1]	SQLF_KTN_NODETYPE	100	Uint16	"nodetype - Instance node type" on page 712

Table 125. Informational Database Manager Configuration Parameters (continued)

Parameter	Token	Token Value	Data Type	Additional Information
release	SQLF_KTN_RELEASE	101	Uint16	"release - Configuration file release level" on page 718

Note:

Please look at the header files `sqlenv.h` and `sqlutil.h` for the valid values and for the definitions used by the configuration parameters.

1. Valid values:

```
SQLF_NT_STANDALONE (0)
SQLF_NT_SERVER (1)
SQLF_NT_REQUESTOR (2)
SQLF_NT_STAND_REQ (3)
SQLF_NT_MPP (4)
SQLF_NT_SATELLITE (5)
```

Database Configuration Parameter Summary

The following table lists the parameters in the database configuration file. When changing the database configuration parameters, consider the detailed information for the parameter.

For some database configuration parameters, changes only take effect when the database is reactivated. In these cases, all applications must first disconnect from the database. (If the database was activated, then it must be deactivated and reactivated.) The changes take effect at the next connection to the database. Other parameters can be changed online; these are called *configurable online configuration parameters*.

Refer to the Database Manager Configuration Parameter Summary section for a description of the "Auto.", "Perf. Impact", "Token", "Token Value", and "Data Type" columns.

The column "Member Cfg." only applies for a DB2 pureScale environment. While all database configuration parameters can be configured globally, the column indicates if a database configuration parameter is able to be configured on a per-member basis. For more information about the database configuration parameters that are configurable on a per-member basis, see DB2 pureScale Feature database configuration parameters.

The **AUTOMATIC** keyword is also supported on the **UPDATE DB CFG** command. In the following example, **database_memory** will be updated to AUTOMATIC and the database manager will use 20000 as a starting value when making further changes to this parameter:

```
db2 update db cfg using for sample using database_memory 20000 automatic
```

Starting with Version 9.5, you can update and reset database configuration parameter values across some or all partitions without having to issue the **db2_all** command, or without having to update or reset each partition individually.

If a database is created by either the **CREATE DATABASE**, or the sqlecrea API, then the Configuration Advisor runs by default to update the database configuration parameters with automatically computed values. If a database is created by either the **CREATE DATABASE** command with the AUTOCONFIGURE APPLY NONE clause added, or the sqlecrea API specifies not to run the Configuration Advisor, then the configuration parameters are set to the default values.

Table 126. Configurable Database Configuration Parameters

Parameter	Cfg. Online	Auto.	Member Cfg.	Perf. Impact	Token	Token Value	Data Type	Additional Information
alt_collate	No	No	No	None	SQLF_DBTN_ALT_COLLATE	809	Uint32	"alt_collate - Alternate collating sequence" on page 741
applheapsz	Yes	Yes	Yes	Medium	SQLF_DBTN_APPLHEAPSZ	51	Uint16	"applheapsz - Application heap size" on page 745
appl_memory	Yes	Yes	Yes	Medium	SQLF_DBTN_APPL_MEMORY	904	Uint64	"appl_memory - Application Memory configuration parameter" on page 744
archretrydelay	Yes	No	No	None	SQLF_DBTN_ ARCHRETRYDELAY	828	Uint16	"archretrydelay - Archive retry delay on error" on page 745
• **auto_ maint** • **auto_ db_backup** • **auto_ tbl_maint** • **auto_ runstats** • **auto_ stats_prof** • **auto_ stmt_stats** • **auto_ stats_views** • **auto_ prof_upd** • **auto_ reorg**	Yes	No	No	Medium	• SQLF_DBTN_AUTO_MAINT • SQLF_DBTN_AUTO_DB_ BACKUP • SQLF_DBTN_AUTO_TBL_ MAINT • SQLF_DBTN_AUTO_ RUNSTATS • SQLF_DBTN_AUTO_STATS_ PROF • SQLF_DBTN_AUTO_STMT_ STATS • SQLF_DBTN_AUTO_STATS_ VIEWS • SQLF_DBTN_AUTO_PROF_ UPD • SQLF_DBTN_AUTO_REORG	• 831 • 833 • 835 • 837 • 839 • 905 • 980 • 844 • 841	Uint32	"auto_maint - Automatic maintenance" on page 746
auto_del_ rec_obj	Yes	No	No	Medium	SQLF_DBTN_AUTO_DEL_ REC_OBJ	912	Uint16	"auto_del_rec_obj - Automated deletion of recovery objects configuration parameter" on page 746
autorestart	Yes	No	No	Low	SQLF_DBTN_AUTO_RESTART	25	Uint16	"autorestart - Auto restart enable" on page 750
auto_reval	Yes	No	Yes	Medium	SQLF_DBTN_AUTO_REVAL	920	Uint16	"auto_reval - Automatic revalidation and invalidation configuration parameter" on page 749
avg_appls	Yes	Yes	Yes	High	SQLF_DBTN_AVG_APPLS	47	Uint16	"avg_appls - Average number of active applications" on page 751
blk_log_dsk_ful	Yes	No	Yes	None	SQLF_DBTN_BLK_LOG_DSK_ FUL	804	Uint16	"blk_log_dsk_ful - Block on log disk full" on page 752
blocknonlogged	Yes	No	Yes	Low	SQLF_DBTN_ BLOCKNONLOGGED	940	Uint16	"blocknonlogged - Block creation of tables that allow non-logged activity" on page 753
catalogcache_sz	Yes	No	Yes	Medium	SQLF_DBTN_ CATALOGCACHE_SZ	56	Uint32	"catalogcache_sz - Catalog cache size" on page 757
chngpgs_thresh	No	No	Yes	High	SQLF_DBTN_CHNGPGS_ THRESH	38	Uint16	"chngpgs_thresh - Changed pages threshold" on page 758
connect_proc	Yes	No	No	None	SQLF_DBTN_CONNECT_PROC	954	char(257)	"connect_proc - Connect procedure name database configuration parameter" on page 761
cur_commit	No	No	Yes	Medium	SQLF_DBTN_CUR_COMMIT	917	Uint32	"cur_commit - Currently committed configuration parameter" on page 678

Table 126. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Member Cfg.	Perf. Impact	Token	Token Value	Data Type	Additional Information
database_memory	Yes	Yes	Yes	Medium	SQLF_DBTN_DATABASE_ MEMORY	803	Uint64	"database_memory - Database shared memory size" on page 762
dbheap	Yes	Yes	Yes	Medium	SQLF_DBTN_DB_HEAP	58	Uint64	"dbheap - Database heap" on page 764
db_mem_thresh	Yes	No	Yes	Low	SQLF_DBTN_DB_MEM_ THRESH	849	Uint16	"db_mem_thresh - Database memory threshold" on page 766
decflt_rounding	No	No	No	None	SQLF_DBTN_DECFLT_ ROUNDING	913	Uint16	"decflt_rounding - Decimal floating point rounding configuration parameter" on page 768
dec_to_char_fmt	Yes	Yes	Yes	Medium	SQLF_DBTN_DEC_TO_ CHAR_FMT	• 0: V95 • 1: NEW	Uint16	"dec_to_char_fmt - Decimal to character function configuration parameter" on page 767
dft_degree	Yes	No	Yes	High	SQLF_DBTN_DFT_DEGREE	301	Sint32	"dft_degree - Default degree" on page 769
dft_extent_sz	Yes	No	No	Medium	SQLF_DBTN_DFT_EXTENT_SZ	54	Uint32	"dft_extent_sz - Default extent size of table spaces" on page 770
dft_loadrec_ses	Yes	No	Yes	Medium	SQLF_DBTN_DFT_LOADREC_ SES	42	Sint16	"dft_loadrec_ses - Default number of load recovery sessions" on page 771
dft_mttb_types	No	No	No	None	SQLF_DBTN_DFT_MTTB_ TYPES	843	Uint32	"dft_mttb_types - Default maintained table types for optimization" on page 771
dft_prefetch_sz	Yes	Yes	No	Medium	SQLF_DBTN_DFT_PREFETCH_ SZ	40	Sint16	"dft_prefetch_sz - Default prefetch size" on page 772
dft_queryopt	Yes	No	Yes	Medium	SQLF_DBTN_DFT_QUERYOPT	57	Sint32	"dft_queryopt - Default query optimization class" on page 773
dft_refresh_age	No	No	No	Medium	SQLF_DBTN_DFT_REFRESH_ AGE	702	char(22)	"dft_refresh_age - Default refresh age" on page 774
dft_schemas_dcc	Yes	No	No	Medium	SQLF_DBTN_DFT_SCHEMAS_ DCC	10115	Uint16	"dft_schemas_dcc - Default data capture on new schemas configuration parameter" on page 774
discover_db	Yes	No	No	Medium	SQLF_DBTN_DISCOVER	308	Uint16	"discover_db - Discover database" on page 776
dlchktime	Yes	No	No	Medium	SQLF_DBTN_DLCHKTIME	9	Uint32	"dlchktime - Time interval for checking deadlock" on page 776
enable_xmlchar	Yes	No	No	None	SQLF_DBTN_ENABLE_ XMLCHAR	853	Uint32	"enable_xmlchar - Enable conversion to XML configuration parameter" on page 777
failarchpath	Yes	No	No	None	SQLF_DBTN_FAILARCHPATH	826	char(243)	"failarchpath - Failover log archive path" on page 777
hadr_local_host	No	No	N/A	None	SQLF_DBTN_HADR_LOCAL_ HOST	811	char(256)	"hadr_local_host - HADR local host name" on page 779
hadr_local_svc	No	No	N/A	None	SQLF_DBTN_HADR_LOCAL_ SVC	812	char(41)	"hadr_local_svc - HADR local service name" on page 779
hadr_peer_ window	No	No	N/A	Low (see Note 3)	SQLF_DBTN_HADR_PEER_ WINDOW	914	Uint32	"hadr_peer_window - HADR peer window configuration parameter" on page 780
hadr_remote_ host	No	No	N/A	None	SQLF_DBTN_HADR_REMOTE_ HOST	813	char(256)	"hadr_remote_host - HADR remote host name" on page 781

Table 126. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Member Cfg.	Perf. Impact	Token	Token Value	Data Type	Additional Information
hadr_remote_inst	No	No	N/A	None	SQLF_DBTN_HADR_REMOTE_INST	815	char(9)	"hadr_remote_inst - HADR instance name of the remote server" on page 781
hadr_remote_svc	No	No	N/A	None	SQLF_DBTN_HADR_REMOTE_SVC	814	char(41)	"hadr_remote_svc - HADR remote service name" on page 781
hadr_replay_delay	Yes	No	N/A	None	SQLF_DBTN_HADR_REPLAY_DELAY	10119	Sint32	"hadr_replay_delay - HADR replay delay configuration parameter" on page 782
hadr_spool_limit	Yes	No	N/A	None	SQLF_DBTN_HADR_SPOOL_LIMIT	10112	Sint32	"hadr_spool_limit - HADR log spool limit configuration parameter" on page 783
hadr_syncmode	No	No	N/A	None	SQLF_DBTN_HADR_SYNCMODE	817	Uint32	"hadr_syncmode - HADR synchronization mode for log write in peer state" on page 784
hadr_target_list	Yes	No	N/A	None	SQLF_DBTN_HADR_TARGET_LIST	10114	char(890)	"hadr_target_list - HADR target list database configuration parameter" on page 785
hadr_timeout	No	No	N/A	None	SQLF_DBTN_HADR_TIMEOUT	816	Uint32	"hadr_timeout - HADR timeout value" on page 787
indexrec[2]	Yes	No	No	Medium	SQLF_DBTN_INDEXREC	30	Uint16	"indexrec - Index re-creation time" on page 697
locklist	Yes	Yes	Yes	High when it affects escalation	SQLF_DBTN_LOCK_LIST	704	Uint64	"locklist - Maximum storage for lock list" on page 790
locktimeout	No	No	Yes	Medium	SQLF_DBTN_LOCKTIMEOUT	34	Sint16	"locktimeout - Lock timeout" on page 792
log_appl_info	No	No	N/A	Low	SQLF_DBTN_LOG_APPL_INFO	10111	Uint32	"log_appl_info - Application information log record database configuration parameter" on page 793
log_ddl_stmts	Yes	No			SQLF_DBTN_LOG_DDL_STMTS	10110	Uint32	"log_ddl_stmts - Log Data Definition Language (DDL) statements database configuration parameter" on page 794
logarchcompr1	Yes	No	No	None	SQLF_DBTN_LOGARCHCOMPR1	10123	char(252)	"logarchcompr1 - Primary archived log file compression configuration parameter" on page 794
logarchcompr2	Yes	No	No	None	SQLF_DBTN_LOGARCHCOMPR2	10124	char(252)	"logarchcompr2 - Secondary archived log file compression configuration parameter" on page 795
logarchmeth1	Yes	No	No	None	SQLF_DBTN_LOGARCHMETH1	822	char(252)	"logarchmeth1 - Primary log archive method" on page 795
logarchmeth2	Yes	No	No	None	SQLF_DBTN_LOGARCHMETH2	823	char(252)	"logarchmeth2 - Secondary log archive method" on page 797
logarchopt1	Yes	No	No	None	SQLF_DBTN_LOGARCHOPT1	824	char(243)	"logarchopt1 - Primary log archive options" on page 798
logarchopt2	Yes	No	No	None	SQLF_DBTN_LOGARCHOPT2	825	char(243)	"logarchopt2 - Secondary log archive options" on page 799
logbufsz	No	No	Yes	High	SQLF_DBTN_LOGBUFSZ	33	Uint16	"logbufsz - Log buffer size" on page 799
logfilsiz	No	No	No	Medium	SQLF_DBTN_LOGFIL_SIZ	92	Uint32	"logfilsiz - Size of log files" on page 800
logindexbuild	Yes	No	Yes	None	SQLF_DBTN_LOGINDEXBUILD	818	Uint32	"logindexbuild - Log index pages created" on page 801

Table 126. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Member Cfg.	Perf. Impact	Token	Token Value	Data Type	Additional Information
logprimary	No	No	No	Medium	SQLF_DBTN_LOGPRIMARY	16	Uint16	"logprimary - Number of primary log files" on page 802
logsecond	Yes	No	No	Medium	SQLF_DBTN_LOGSECOND	17	Uint16	"logsecond - Number of secondary log files" on page 803
max_log	Yes	Yes	Yes		SQLF_DBTN_MAX_LOG	807	Uint16	"max_log - Maximum log per transaction" on page 805
maxappls	Yes	Yes	Yes	Medium	SQLF_DBTN_MAXAPPLS	6	Uint16	"maxappls - Maximum number of active applications" on page 806
maxfilop	Yes	No	Yes	Medium	SQLF_DBTN_MAXFILOP	3	Uint16	"maxfilop - Maximum database files open per database" on page 807
maxlocks	Yes	Yes	Yes	High when it affects escala-tion	SQLF_DBTN_MAXLOCKS	15	Uint16	"maxlocks - Maximum percent of lock list before escalation" on page 808
min_dec_div_3	No	No	No	High	SQLF_DBTN_MIN_DEC_DIV_3	605	Sint32	"min_dec_div_3 - Decimal division scale to 3" on page 809
mincommit	Yes	No	Yes	High	SQLF_DBTN_MINCOMMIT	32	Uint16	"mincommit - Number of commits to group" on page 810
mirrorlogpath	No	No	No	Low	SQLF_DBTN_ MIRRORLOGPATH	806	char(242)	"mirrorlogpath - Mirror log path" on page 811
mon_act_metrics	Yes	No	Yes	Medium	SQLF_DBTN_MON_ ACT_METRICS	931	Uint16	"mon_act_metrics - Monitoring activity metrics configuration parameter" on page 813
mon_deadlock	Yes	No	Yes	Medium	SQLF_DBTN_MON_ DEADLOCK	934	Uint16	"mon_deadlock - Monitoring deadlock configuration parameter" on page 814
mon_locktimeout	Yes	No	Yes	Medium	SQLF_DBTN_MON_ LOCKTIMEOUT	933	Uint16	"mon_locktimeout - Monitoring lock timeout configuration parameter" on page 815
mon_lockwait	Yes	No	Yes	Medium	SQLF_DBTN_MON_ LOCKWAIT	935	Uint16	"mon_lockwait - Monitoring lock wait configuration parameter" on page 816
mon_lw_thresh	Yes	No	Yes	Medium	SQLF_DBTN_MON_ LW_THRESH	936	Uint32	"mon_lw_thresh - Monitoring lock wait threshold configuration parameter" on page 816
mon_lck_msg_lvl	Yes	No	Yes	None	SQLF_DBTN_MON_ LCK_MSG_LVL	951	Uint16	"mon_lck_msg_lvl - Monitoring lock event notification messages configuration parameter" on page 817
mon_obj_metrics	Yes	No	Yes	Medium	SQLF_DBTN_MON_ OBJ_METRICS	937	Uint16	"mon_obj_metrics - Monitoring object metrics configuration parameter" on page 817
mon_pkglist_sz	Yes	No	Yes	Low	SQLF_DBTN_MON_ PKGLIST_SZ	950	Uint32	"mon_pkglist_sz - Monitoring package list size configuration parameter" on page 819
mon_req_metrics	Yes	No	Yes	Medium	SQLF_DBTN_MON_ REQ_METRICS	930	Uint16	"mon_req_metrics - Monitoring request metrics configuration parameter" on page 820

Table 126. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Member Cfg.	Perf. Impact	Token	Token Value	Data Type	Additional Information
mon_uow_data	Yes	No	Yes	Medium	SQLF_DBTN_MON_ UOW_DATA	932	Uint16	"mon_uow_data - Monitoring unit of work events configuration parameter" on page 821
mon_uow_execlist	Yes	No	Yes	Medium	SQLF_DBTN_MON_ UOW_EXECLIST	957	Uint16	"mon_uow_execlist - Monitoring unit of work events with executable list configuration parameter" on page 822
mon_uow_pkglist	Yes	No	Yes	Medium	SQLF_DBTN_MON_ UOW_PKGLIST	956	Uint16	"mon_uow_pkglist - Monitoring unit of work events with package list configuration parameter" on page 822
newlogpath	No	No	No	Low	SQLF_DBTN_NEWLOGPATH	20	char(242)	"newlogpath - Change the database log path" on page 823
num_db_backups	Yes	No	No	None	SQLF_DBTN_NUM_ DB_BACKUPS	601	Uint16	"num_db_backups - Number of database backups" on page 825
num_freqvalues	Yes	No	No	Low	SQLF_DBTN_NUM_ FREQVALUES	36	Uint16	"num_freqvalues - Number of frequent values retained" on page 825
num_iocleaners	No	Yes	Yes	High	SQLF_DBTN_NUM_ IOCLEANERS	37	Uint16	"num_iocleaners - Number of asynchronous page cleaners" on page 826
num_ioservers	No	Yes	Yes	High	SQLF_DBTN_NUM_ IOSERVERS	39	Uint16	"num_ioservers - Number of I/O servers" on page 828
num_log_span	Yes	Yes	Yes		SQLF_DBTN_NUM_ LOG_SPAN	808	Uint16	"num_log_span - Number log span" on page 828
num_quantiles	Yes	No	Yes	Low	SQLF_DBTN_NUM_ QUANTILES	48	Uint16	"num_quantiles - Number of quantiles for columns" on page 829
numarchretry	Yes	No	Yes	None	SQLF_DBTN_ NUMARCHRETRY	827	Uint16	"numarchretry - Number of retries on error" on page 830
overflowlogpath	Yes	No	No	Medium	SQLF_DBTN_ OVERFLOWLOGPATH	805	char(242)	"overflowlogpath - Overflow log path" on page 832
pckcachesz	Yes	Yes	Yes	High	SQLF_DBTN_PCKCACHE_SZ	505	Uint32	"pckcachesz - Package cache size" on page 833
rec_his_retentn	No	No	No	None	SQLF_DBTN_REC_HIS_ RETENTN	43	Sint16	"rec_his_retentn - Recovery history retention period" on page 835
section_actuals	Yes	No	No	High	SQLF_DBTN_ SECTION_ACTUALS	952	Uint64	"section_actuals - Section actuals configuration parameter" on page 837
self_tuning_mem	Yes	No	Yes	High	SQLF_DBTN_SELF_ TUNING_MEM	848	Uint16	"self_tuning_mem- Self-tuning memory" on page 837
seqdetect	Yes	No	No	High	SQLF_DBTN_SEQDETECT	41	Uint16	"seqdetect - Sequential detection and readahead flag" on page 839
sheapthres_shr	Yes	Yes	Yes	High	SQLF_DBTN_SHEAPTHRES_ SHR	802	Uint32	"sheapthres_shr - Sort heap threshold for shared sorts" on page 840
smtp_server	Yes	No	Yes	None	SQLF_DBTN_SMTP_SERVER	926	char [] (String)	"smtp_server - SMTP server" on page 841
softmax	No	No	No	Medium	SQLF_DBTN_SOFTMAX	5	Uint16	"softmax - Recovery range and soft checkpoint interval" on page 842
sortheap	Yes	Yes	Yes	High	SQLF_DBTN_ SORT_HEAP	52	Uint32	"sortheap - Sort heap size" on page 843

Table 126. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Member Cfg.	Perf. Impact	Token	Token Value	Data Type	Additional Information
sql_ccflags	Yes	No	Yes	None	SQLF_DBTN_ SQL_CCFLAGS	927	char(1023)	"sql_ccflags - Conditional compilation flags" on page 845
stat_heap_sz	Yes	Yes	Yes	Low	SQLF_DBTN_ STAT_HEAP_SZ	45	Uint32	"stat_heap_sz - Statistics heap size" on page 845
stmt_conc	Yes	No	Yes	Medium	SQLF_DBTN_ STMT_CONC	919	Uint32	"stmt_conc - Statement concentrator configuration parameter" on page 846
stmtheap	Yes	Yes	Yes	Medium	SQLF_DBTN_ STMT_HEAP	821	Uint32	"stmtheap - Statement heap size" on page 847
systime_period _adj	Yes	No	No	None	SQLF_DBTN_SYSTIME_ PERIOD_ADJ	955	Uint16	"systime_period_adj - Adjust temporal SYSTEM_TIME period database configuration parameter" on page 849
trackmod	No	No	No	Low	SQLF_DBTN_TRACKMOD	703	Uint16	"trackmod - Track modified pages enable" on page 850
tsm_mgmtclass	Yes	No	Yes	None	SQLF_DBTN_TSM_ MGMTCLASS	307	char(30)	"tsm_mgmtclass - Tivoli Storage Manager management class" on page 850
tsm_nodename	Yes	No	Yes	None	SQLF_DBTN_TSM_ NODENAME	306	char(64)	"tsm_nodename - Tivoli Storage Manager node name" on page 851
tsm_owner	Yes	No	Yes	None	SQLF_DBTN_TSM_ OWNER	305	char(64)	"tsm_owner - Tivoli Storage Manager owner name" on page 851
tsm_password	Yes	No	Yes	None	SQLF_DBTN_TSM_ PASSWORD	501	char(64)	"tsm_password - Tivoli Storage Manager password" on page 852
util_heap_sz	Yes	No	Yes	Low	SQLF_DBTN_UTIL_ HEAP_SZ	55	Uint32	"util_heap_sz - Utility heap size" on page 853
vendoropt	Yes	No	No	None	SQLF_DBTN_VENDOROPT	829	char(242)	"vendoropt - Vendor options" on page 853<
wlm_collect_ int	Yes	No	Yes	Low	SQLF_DBTN_WLM_ COLLECT_INT	907	Sint32	"wlm_collect_int - Workload management collection interval configuration parameter" on page 854

Table 126. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Member Cfg.	Perf. Impact	Token	Token Value	Data Type	Additional Information

Note:

Please look at the header files `sqlenv.h` and `sqlutil.h` for the valid values and for the definitions used by the configuration parameters.

1. The bits of SQLF_DBTN_AUTONOMIC_SWITCHES indicate the default settings for a number of auto-maintenance configuration parameters. The individual bits making up this composite parameter are:

```
Default => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint
Bit 2 off (xxxx xxxx xxxx xx0x): auto_db_backup
Bit 3 on  (xxxx xxxx xxxx x1xx): auto_tbl_maint
Bit 4 on  (xxxx xxxx xxxx 1xxx): auto_runstats
Bit 5 off (xxxx xxxx xxx0 xxxx): auto_stats_prof
Bit 6 off (xxxx xxxx xx0x xxxx): auto_prof_upd
Bit 7 off (xxxx xxxx x0xx xxxx): auto_reorg
Bit 8 off (xxxx xxxx 0xxx xxxx): auto_storage
Bit 9 off (xxxx xxx0 xxxx xxxx): auto_stmt_stats
0 0 0 D

Maximum => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint
Bit 2 off (xxxx xxxx xxxx xx1x): auto_db_backup
Bit 3 on  (xxxx xxxx xxxx x1xx): auto_tbl_maint
Bit 4 on  (xxxx xxxx xxxx 1xxx): auto_runstats
Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof
Bit 6 off (xxxx xxxx xx1x xxxx): auto_prof_upd
Bit 7 off (xxxx xxxx x1xx xxxx): auto_reorg
Bit 8 off (xxxx xxxx 1xxx xxxx): auto_storage
Bit 9 off (xxxx xxx1 xxxx xxxx): auto_stmt_stats
0 1 F F
```

2. Valid values:

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
SQLF_INX_REC_RESTART (2)
SQLF_INX_REC_RESTART_NO_REDO (3)
SQLF_INX_REC_ACCESS_NO_REDO (4)
```

3. If you set the **hadr_peer_window** parameter to a nonzero time value, then the primary database might seem to hang on transactions when it is in disconnected peer state, because it is waiting for confirmation from the standby database even though it is not connected to the standby database.

Table 127. Informational Database Configuration Parameters

Parameter	Token	Token Value	Data Type	Additional Information
backup_pending	SQLF_DBTN_BACKUP_PENDING	112	Uint16	"backup_pending - Backup pending indicator" on page 752
codepage	SQLF_DBTN_CODEPAGE	101	Uint16	"codepage - Code page for the database" on page 759
codeset	SQLF_DBTN_CODESET	120	char(9)[1]	"codeset - Codeset for the database" on page 759
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)	"collate_info - Collating information" on page 760
country/region	SQLF_DBTN_COUNTRY	100	Uint16	"country/region - Database territory code" on page 762
database_consistent	SQLF_DBTN_CONSISTENT	111	Uint16	"database_consistent - Database is consistent" on page 762
database_level	SQLF_DBTN_DATABASE_LEVEL	124	Uint16	"database_level - Database release level" on page 762
hadr_db_role	SQLF_DBTN_HADR_DB_ROLE	810	Uint32	"hadr_db_role - HADR database role" on page 778
log_retain_status	SQLF_DBTN_LOG_RETAIN_STATUS	114	Uint16	"log_retain_status - Log retain status indicator" on page 794
loghead	SQLF_DBTN_LOGHEAD	105	char(12)	"loghead - First active log file" on page 801
logpath	SQLF_DBTN_LOGPATH	103	char(242)	"logpath - Location of log files" on page 802
multipage_alloc	SQLF_DBTN_MULTIPAGE_ALLOC	506	Uint16	"multipage_alloc - Multipage file allocation enabled" on page 823
numsegs	SQLF_DBTN_NUMSEGS	122	Uint16	"numsegs - Default number of SMS containers" on page 831

Table 127. Informational Database Configuration Parameters (continued)

Parameter	Token	Token Value	Data Type	Additional Information
pagesize	SQLF_DBTN_PAGESIZE	846	Uint32	"pagesize - Database default page size" on page 833
release	SQLF_DBTN_RELEASE	102	Uint16	"release - Configuration file release level" on page 718
restore_pending	SQLF_DBTN_RESTORE_PENDING	503	Uint16	"restore_pending - Restore pending" on page 836
restrict_access	SQLF_DBTN_RESTRICT_ACCESS	852	Sint32	"restrict_access - Database has restricted access configuration parameter" on page 836
rollfwd_pending	SQLF_DBTN_ROLLFWD_PENDING	113	Uint16	"rollfwd_pending - Roll forward pending indicator" on page 836
suspend_io	SQLF_DBTN_SUSPEND_IO	953	Uint16	"suspend_io - Database I/O operations state configuration parameter" on page 848
territory	SQLF_DBTN_TERRITORY	121	char(5)[2]	"territory - Database territory" on page 850
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16	"user_exit_status - User exit status indicator" on page 852

Note:

1. char(17) on HP-UX, Linux and Solaris operating systems.
2. char(33) on HP-UX, Linux and Solaris operating systems.

DB2 Administration Server (DAS) Configuration Parameter Summary

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 .

Table 128. DAS Configuration Parameters

Parameter	Parameter Type	Additional Information
authentication	Configurable	"authentication - Authentication type DAS" on page 855
contact_host	Configurable Online	"contact_host - Location of contact list" on page 856
das_codepage	Configurable Online	"das_codepage - DAS code page" on page 856
das_territory	Configurable Online	"das_territory - DAS territory" on page 857
dasadm_group	Configurable	"dasadm_group - DAS administration authority group name" on page 857
db2system	Configurable Online	"db2system - Name of the DB2 server system" on page 858
discover	Configurable Online	"discover - DAS discovery mode" on page 859
exec_exp_task	Configurable	"exec_exp_task - Execute expired tasks" on page 860
jdk_64_path	Configurable Online	"jdk_64_path - 64-Bit Software Developer's Kit for Java installation path DAS" on page 789
jdk_path	Configurable Online	"jdk_path - Software Developer's Kit for Java installation path DAS" on page 860
sched_enable	Configurable	"sched_enable - Scheduler mode" on page 860
sched_userid	Informational	"sched_userid - Scheduler user ID" on page 861
smtp_server	Configurable Online	"smtp_server - SMTP server" on page 861
toolscat_db	Configurable	"toolscat_db - Tools catalog database" on page 862
toolscat_inst	Configurable	"toolscat_inst - Tools catalog database instance" on page 862
toolscat_schema	Configurable	"toolscat_schema - Tools catalog database schema" on page 862

Ingest Utility Configuration Parameter Summary

Table 129. Ingest Utility Configuration Parameters

Parameter	Parameter Type	Additional Information
commit_count	Configurable	"commit_count - Commit count configuration parameter" on page 873
commit_period	Configurable	"commit_period - Commit period configuration parameter" on page 874
num_flushers _per_partition	Configurable	"num_flushers_per_partition - Number of flushers per database partition configuration parameter" on page 875
num_formatters	Configurable	"num_formatters - Number of formatters configuration parameter" on page 875
pipe_timeout	Configurable	"pipe_timeout - Pipe timeout configuration parameter" on page 875
retry_count	Configurable	"retry_count - Retry count configuration parameter" on page 876
retry_period	Configurable	"retry_period - Retry period configuration parameter" on page 876
shm_max_size	Configurable	"shm_max_size - Maximum size of shared memory configuration parameter" on page 877

Configuration parameter section headings

Each of the configuration parameter descriptions contain some or all of the following section headings, as applicable. In some cases they are mutually exclusive, for example, valid values are not needed if the [range] is specified. In most cases, these headings are self-explanatory.

Table 130. Description of the configuration parameter section headings

Section heading	Description and possible values
Configuration type	Possible values are: • Database manager • Database • DB2 Administration Server
Applies to	If applicable, lists the data server types that the configuration parameter applies to. Possible values are: • Client • Database server with local and remote clients • Database server with local clients • DB2 Administration Server • OLAP functions • Partitioned database server with local and remote clients • Partitioned database server with local and remote clients when federation is enabled. • Satellite database server with local clients
Parameter type	Possible values are: • Configurable (the database manager must be restarted to have the changes take effect) • Configurable online (can be dynamically updated online without having to restart the database manager) • Informational (values are for your information only and cannot be updated) • Configurable by member in a DB2 pureScale environment
Default [range]	If applicable, lists the default value and the possible ranges, including NULL values or automatic settings. If the range differs by platform, then the values are listed by platform or platform type, for example, 32-bit or 64-bit platforms. Note that in most cases the default value is not listed as part of the range.

Table 130. Description of the configuration parameter section headings (continued)

Section heading	Description and possible values
Unit of measure	If applicable, lists the unit of measure. Possible values are: • Bytes • Counter • Megabytes per second • Milliseconds • Minutes • Pages (4 KB) • Percentage • Seconds
Valid values	If applicable, lists the valid value. This heading is mutually exclusive with the default [range] heading.
Examples	If applicable, lists examples.
Propagation class	If applicable, possible values are: • Immediate • Statement boundary • Transaction boundary • Connection
When allocated	If applicable, indicates when the configuration parameter is allocated by the database manager.
When freed	If applicable, indicates when the configuration parameter is freed by the database manager.
Restrictions	If applicable, lists any restrictions that apply to the configuration parameter.
Limitations	If applicable, lists any limitations that apply to the configuration parameter.
Recommendations	If applicable, lists any recommendations that apply to the configuration parameter.
Usage notes	If applicable, lists any usage notes that apply to the configuration parameter.

Configuration parameters that affect the number of agents

There are a number of database manager configuration parameters related to database agents and how they are managed.

The following database manager configuration parameters determine how many database agents are created and how they are managed:

- Agent Pool Size (**num_poolagents**): The total number of idle agents to pool that are kept available in the system. The default value for this parameter is `100, AUTOMATIC`.
- Initial Number of Agents in Pool (**num_initagents**): When the database manager is started, a pool of worker agents is created based on this value. This speeds up performance for initial queries. The worker agents all begin as idle agents.
- Maximum Number of Connections (**max_connections**): specifies the maximum number of connections allowed to the database manager system on each database partition.
- Maximum Number of Coordinating Agents (**max_coordagents**): For partitioned database environments and environments with intrapartition parallelism enabled when connection concentrator is enabled, this value limits the number of coordinating agents.

Configuration parameters that affect query optimization

Several configuration parameters affect the access plan chosen by the SQL or XQuery compiler. Many of these are appropriate to a single-partition database environment and some are only appropriate to a partitioned database environment. Assuming a homogeneous partitioned database environment, where the hardware is the same, the values used for each parameter should be the same on all database partitions.

Note: When you change a configuration parameter dynamically, the optimizer might not read the changed parameter values immediately because of older access plans in the package cache. To reset the package cache, execute the FLUSH PACKAGE CACHE statement.

In a federated system, if the majority of your queries access nicknames, evaluate the types of queries that you send before you change your environment. For example, in a federated database the buffer pool does not cache pages from data sources, which are the database management systems and data within the federated system. For this reason, increasing the size of the buffer does not guarantee that the optimizer will consider additional access-plan alternatives when it chooses an access plan for queries that contain nicknames. However, the optimizer might decide that local materialization of data source tables is the least-cost route or a necessary step for a sort operation. In that case, increasing the resources available might improve performance.

The following configuration parameters or factors affect the access plan chosen by the SQL or XQuery compiler:

- The size of the buffer pools that you specified when you created or altered them.

  When the optimizer chooses the access plan, it considers the I/O cost of fetching pages from disk to the buffer pool and estimates the number of I/Os required to satisfy a query. The estimate includes a prediction of buffer pool usage, because additional physical I/Os are not required to read rows in a page that is already in the buffer pool.

  The optimizer considers the value of the NPAGES column in the SYSCAT.BUFFERPOOLS system catalog tables and, in partitioned database environments, the SYSCAT.BUFFERPOOLDBPARTITIONS system catalog tables.

  The I/O costs of reading the tables can have an impact on:

  – How two tables are joined
  – Whether an unclustered index will be used to read the data
- Default Degree (**dft_degree**)

  The **dft_degree** configuration parameter specifies parallelism by providing a default value for the CURRENT DEGREE special register and the **DEGREE** bind option. A value of one (1) means no intrapartition parallelism. A value of minus one (-1) means the optimizer determines the degree of intrapartition parallelism based on the number of processors and the type of query.

  Note: Intra-parallel processing does not occur unless you enable it by setting the **intra_parallel** database manager configuration parameter.
- Default Query Optimization Class (**dft_queryopt**)

  Although you can specify a query optimization class when you compile SQL or XQuery queries, you can also set a default query optimization class.
- Average Number of Active Applications (**avg_appls**)

The optimizer uses the **avg_appls** parameter to help estimate how much of the buffer pool might be available at run time for the access plan chosen. Higher values for this parameter can influence the optimizer to choose access plans that are more conservative in buffer pool usage. If you specify a value of 1, the optimizer considers that the entire buffer pool will be available to the application.

- Sort Heap Size (**sortheap**)

  If the rows to be sorted occupy more than the space available in the sort heap, several sort passes are performed, where each pass sorts a subset of the entire set of rows. Each sort pass is stored in a system temporary table in the buffer pool, which might be written to disk. When all the sort passes are complete, these sorted subsets are merged into a single sorted set of rows. A sort that does not require a system temporary table to store the list of data always results in better performance and is used if possible.

  When choosing an access plan, the optimizer estimates the cost of the sort operations, including evaluating whether a sort can be read in a single, sequential access, by:

  – Estimating the amount of data to be sorted
  – Looking at the **sortheap** parameter to determine if there is enough space to read a sort in a single, sequential access.

- Maximum Storage for Lock List (**locklist**) and Maximum Percent of Lock List Before Escalation (**maxlocks**)

  When the isolation level is repeatable read (RR), the optimizer considers the values of the **locklist** and **maxlocks** parameters to determine whether row level locks might be escalated to a table level lock. If the optimizer estimates that lock escalation will occur for a table access, then it chooses a table level lock for the access plan, instead of requiring additional processing time caused by lock escalation during the query execution.

- CPU Speed (**cpuspeed**)

  The optimizer uses the CPU speed to estimate the cost of performing certain operations. CPU cost estimates and various I/O cost estimates help select the best access plan for a query.

  The CPU speed of a machine can have a significant influence on the access plan chosen. This configuration parameter is automatically set to an appropriate value when the database is installed or upgraded. Do not adjust this parameter unless you are modelling a production environment on a test system or assessing the impact of a hardware change. Using this parameter to model a different hardware environment allows you to find out the access plans that might be chosen for that environment. To have the database manager recompute the value of this automatic configuration parameter, set it to -1.

- Statement Heap Size (**stmtheap**)

  Although the size of the statement heap does not influence the optimizer in choosing different access paths, it can affect the amount of optimization performed for complex SQL or XQuery statements.

  If the **stmtheap** parameter is not set large enough, you might receive a warning indicating that there is not enough memory available to process the statement. For example, SQLCODE +437 (SQLSTATE 01602) might indicate that the amount of optimization that has been used to compile a statement is less than the amount that you requested.

- Communications Bandwidth (**comm_bandwidth**)

Communications bandwidth is used by the optimizer to determine access paths. The optimizer uses the value in this parameter to estimate the cost of performing certain operations between the database partition servers in a partitioned database environment.

- Application Heap Size (**applheapsz**)

  Large schemas require sufficient space in the application heap.

Configuration parameters that affect the DB2 pureScale Feature

In a DB2 pureScale environment, database configuration parameters are designated as either *global* database configuration parameters or *per-member* database configuration parameters. This distinction allows a DB2 pureScale Feature instance to benefit from global database configuration consistency, though accommodating for differences between members.

Global database configuration parameters

Global database configuration parameters share a single, consistent value for all members in a DB2 instance. An update to a value of a global parameter applies globally across all members, regardless of which member issues the update. Global database configuration consistency ensures all members in a DB2 instance access and manipulate the same set of data homogeneously.

Per-member database configuration parameters

Per-member database configuration parameters can have different values between each member in a DB2 instance. By default, an update to a value of a per-member parameter applies globally across all members, unless that update is specified for a specific member only. Any member in the DB2 instance can issue an update. Per-member database configuration parameters in the DB2 pureScale Feature are intended for use when a non-homogenous environment is wanted.

For a database, where the available resources on each member are different, customizing database configurations on each member can optimize database performance. For example, differences in the available memory on each member can benefit from unique database configurations on each member.

A database where members are mapped to applications of differing function can benefit from customized database configurations on each member.

Updating database configuration parameters in a DB2 pureScale environment

You can update database configuration parameters in a DB2 pureScale instance using the DB2 command line processor (CLP) or using DB2 configuration APIs. The use of the **MEMBER** clause is optional and is only intended for use with per-member database configuration parameters.

Updating global database configuration parameters

Global database configuration parameters are stored in a single database configuration file. The updating member is responsible for updating the database configuration file. Updates to global parameters fail only if the updating member is unable to write to the global configuration file.

Note: No rollback is required when a member fails to write to the configuration file because all members within the DB2 pureScale instance are in a consistent state.

If you attempt to update a global database configuration parameter using the **MEMBER** clause you receive an error (SQL5125N).

Updating per-member database configuration parameters

Members within a DB2 pureScale instance can be configured with different values for the same configuration parameter. Use the **MEMBER** clause to specify an update to a single member within an instance. If you omit the **MEMBER** clause the change is applied across all members in the instance.

The following command updates the **util_heap_sz** parameter for MEMBER 2 to a value of 5000:

```
UPDATE DATABASE CONFIGURATION FOR WSDB MEMBER 2 USING UTIL_HEAP_SZ 5000
```

Only MEMBER 2 is updated with a value of 5000 for **util_heap_sz**. If you omit the **MEMBER** clause then **util_heap_sz** is updated to the value 5000 for all members across the DB2 pureScale instance.

This command can be issued by any member within the DB2 pureScale instance. In the event MEMBER 2 goes down or becomes inactive, the configuration update might or might not fail. Every member in a DB2 pureScale instance can write to the configuration file of another member, so the update fails only if the updating member is unable to successfully write to the configuration file of MEMBER 2.

Note: If the **MEMBER** clause is not specified and the updating member attempts and fails to update the configuration file for MEMBER 2 then the update is rolled back across the entire DB2 pureScale instance. If the rollback fails as well, the configuration files might be in an inconsistent state.

For database configuration parameters that can be updated when the database is online, no additional measures are taken to ensure that cached and in-memory values are consistent across a DB2 pureScale instance.

Note: The values for global configuration parameters are stored in the global configuration file under the partition-global directory; values for per-member configuration parameters are stored in each member's member-specific directory.

DB2 pureScale Feature configuration parameters

For the IBM DB2 pureScale Feature, there are several configuration parameters to support members and the cluster caching facility, also known as CF. These database configuration parameters divide into *global* and *per-member* parameters.

DB2 pureScale Feature database manager configuration parameters

The following table lists the database manager configuration parameters for the cluster caching facility. The database manager configuration parameters apply to instance level configuration.

Table 131. Summary of the DB2 pureScale Feature database manager configuration parameters

Parameter name	Details
cf_diaglevel	Specifies the level of diagnostic errors that are recorded in the `cfdiag.log` file.

Table 131. Summary of the DB2 pureScale Feature database manager configuration parameters (continued)

Parameter name	Details
cf_diagpath	Specifies the directory that contains the CF diagnostic log file that are generated based on the value of the **cf_diaglevel** parameter.
cf_mem_size	Determines the total memory used by the CF. The value must be less than the amount of physical memory on the CF host.
cf_num_conns	Determines the initial size of the CF connection pool.
cf_num_workers	Determines how many worker threads are started by the CF server. The value is typically less than or equal to the number of CPUs on the CF server.

DB2 pureScale Feature database configuration parameters

The following table lists the database configuration parameters for the cluster caching facility. The database configuration parameters apply to database level configuration.

Note: The CF database configuration parameter cf_db_mem_sz applies at a database wide level. The remaining CF database configuration parameters apply to the structure level and have an upper limit determined by cf_db_mem_sz. For more information about the relationship between database configuration parameters, see the topic "Configuring cluster caching facility memory for a database."

Table 132. Summary of the DB2 pureScale Feature global database configuration parameters

Parameter name	Details
cf_catchup_trgt	Determines the target time in minutes for completing the catch up to bring a newly added or newly restarted CF into peer state with an existing primary CF.
cf_db_mem_sz	Determines the total CF memory limit for the current database.
cf_gbp_sz	Determines the total memory size allocated by the CF for the group buffer pool for the current database.
cf_lock_sz	Determines the total memory size allocated by the CF for the lock structure for the current database.
cf_sca_sz	Determines the total memory size allocated by the CF for the shared communication area for the current database.

The following DB2 database configuration parameters are designated as *global* database configuration parameters for the DB2 pureScale Feature. For more information about global database configuration parameters see the topic "DB2 pureScale Feature database configuration parameters."

Table 133. Configurable global database configuration parameters

Parameter	Additional Information
alt_collate	**alt_collate - Alternate collating sequence configuration parameter**
archretrydelay	**archretrydelay - Archive retry delay on error configuration parameter**
auto_del_rec_obj	auto_del_rec_obj - Automated deletion of recovery objects configuration parameter

Table 133. Configurable global database configuration parameters (continued)

Parameter	Additional Information
• auto_maint • auto_db_backup • auto_tbl_maint • auto_runstats • auto_stats_prof • auto_stmt_stats • auto_prof_upd • auto_reorg	auto_maint - Automatic maintenance configuration parameter
autorestart	autorestart - Auto restart enable configuration parameter
decflt_rounding	decflt_rounding - Decimal floating point rounding configuration parameter
dft_extent_sz	dft_extent_sz - Default extent size of table spaces configuration parameter
dft_mttb_types	dft_mttb_types - Default maintained table types for optimization configuration parameter
dft_prefetch_sz	dft_prefetch_sz - Default prefetch size configuration parameter
dft_refresh_age	dft_refresh_age - Default refresh age configuration parameter
dft_sqlmathwarn	dft_sqlmathwarn - Continue upon arithmetic exceptions configuration parameter
discover_db	discover_db - Discover database configuration parameter
dlchktime	dlchktime - Time interval for checking deadlock configuration parameter
enable_xmlchar	enable_xmlchar - Enable conversion to XML configuration parameter
failarchpath	failarchpath - Failover log archive path configuration parameter
indexrec	indexrec - Index re-creation time configuration parameter
locktimeout	locktimeout - Lock timeout configuration parameter
logarchmeth1	logarchmeth1 - Primary log archive method configuration parameter
logarchmeth2	logarchmeth2 - Secondary log archive method configuration parameter
logarchopt1	logarchopt1 - Primary log archive options configuration parameter
logarchopt2	logarchopt2 - Secondary log archive options configuration parameter
logfilsiz	logfilsiz - Size of log files configuration parameter
logprimary	logprimary - Number of primary log files configuration parameter
logsecond	logsecond - Number of secondary log files configuration parameter
min_dec_div_3	min_dec_div_3 - Decimal division scale to 3 configuration parameter
mirrorlogpath	mirrorlogpath - Mirror log path configuration parameter
mon_act_metrics	mon_act_metrics - Monitoring activity metrics configuration parameter
mon_deadlock	mon_deadlock - Monitoring deadlock configuration parameter
mon_locktimeout	mon_locktimeout - Monitoring lock timeout configuration parameter
mon_lockwait	mon_lockwait - Monitoring lock wait configuration parameter
mon_lw_thresh	mon_lw_thresh - Monitoring lock wait threshold configuration parameter
mon_obj_metrics	mon_obj_metrics - Monitoring object metrics configuration parameter
mon_req_metrics	mon_req_metrics - Monitoring request metrics configuration parameter
mon_uow_data	mon_uow_data - Monitoring unit of work events configuration parameter
newlogpath	newlogpath - Change the database log path configuration parameter
num_db_backups	num_db_backups - Number of database backups configuration parameter

Table 133. Configurable global database configuration parameters (continued)

Parameter	Additional Information
num_freqvalues	num_freqvalues - Number of frequent values retained configuration parameter
numarchretry	numarchretry - Number of retries on error configuration parameter
overflowlogpath	overflowlogpath - Overflow log path configuration parameter
rec_his_retentn	rec_his_retentn - Recovery history retention period configuration parameter
seqdetect	seqdetect - Sequential detection flag configuration parameter
softmax	softmax - Recovery range and soft checkpoint interval configuration parameter
trackmod	trackmod - Track modified pages enable configuration parameter
vendoropt	vendoropt - Vendor options configuration parameter

The following DB2 database configuration parameters are designated as *per-member* database configuration parameters for the DB2 pureScale Feature. For more information about per-member database configuration parameters see the topic "DB2 pureScale Feature database configuration parameters."

Table 134. Configurable Per-member Database Configuration Parameters

Parameter	Additional Information
applheapsz	applheapsz - Application heap size configuration parameter
appl_memory	appl_memory - Application Memory configuration parameter
auto_reval	
avg_appls	avg_appls - Average number of active applications configuration parameter
blk_log_dsk_ful	blk_log_dsk_ful - Block on log disk full configuration parameter
blocknonlogged	blocknonlogged - Block creation of tables that allow non-logged activity configuration parameter
catalogcache_sz	catalogcache_sz - Catalog cache size configuration parameter
chngpgs_thresh	chngpgs_thresh - Changed pages threshold configuration parameter
connect_proc	connect_proc - Connect procedure name database configuration parameter
cur_commit	cur_commit - Currently committed configuration parameter
database_memory	database_memory - Database shared memory size configuration parameter
dbheap	dbheap - Database heap configuration parameter
db_mem_thresh	db_mem_thresh - Database memory threshold configuration parameter
dec_to_char_fmt	dec_to_char_fmt - Decimal to character function configuration parameter
dft_degree	dft_degree - Default degree configuration parameter
dft_loadrec_ses	dft_loadrec_ses - Default number of load recovery sessions configuration parameter
dft_queryopt	dft_queryopt - Default query optimization class configuration parameter
discover_db	discover_db - Discover database configuration parameter
hadr_local_host	hadr_local_host - HADR local host name configuration parameter
hadr_local_svc	hadr_local_svc - HADR local service name configuration parameter
hadr_peer_ window	hadr_peer_window - HADR peer window configuration parameter
hadr_remote_host	hadr_remote_host - HADR remote host name configuration parameter
hadr_remote_inst	hadr_remote_inst - HADR instance name of the remote server configuration parameter
hadr_remote_svc	hadr_remote_svc - HADR remote service name configuration parameter
hadr_syncmode	hadr_syncmode - HADR synchronization mode for log write in peer state configuration parameter
hadr_timeout	hadr_timeout - HADR timeout value configuration parameter

Table 134. Configurable Per-member Database Configuration Parameters (continued)

Parameter	Additional Information
locklist	locklist - Maximum storage for lock list configuration parameter
locktimeout	locktimeout - Lock timeout configuration parameter
logbufsz	logbufsz - Log buffer size configuration parameter
logindexbuild	logindexbuild - Log index pages created configuration parameter
max_log	max_log - Maximum log per transaction configuration parameter
maxappls	maxappls - Maximum number of active applications configuration parameter
maxfilop	maxfilop - Maximum database files open per application configuration parameter
maxlocks	maxlocks - Maximum percent of lock list before escalation configuration parameter
mincommit	mincommit - Number of commits to group configuration parameter
mon_act_metrics	mon_act_metrics - Monitoring activity metrics configuration parameter
mon_deadlock	mon_deadlock - Monitoring deadlock configuration parameter
mon_locktimeout	mon_locktimeout - Monitoring lock timeout configuration parameter
mon_lockwait	mon_lockwait - Monitoring lock wait configuration parameter
mon_lw_thresh	mon_lw_thresh - Monitoring lock wait threshold configuration parameter
mon_obj_metrics	mon_obj_metrics - Monitoring object metrics configuration parameter
mon_req_metrics	mon_req_metrics - Monitoring request metrics configuration parameter
mon_uow_data	mon_uow_data - Monitoring unit of work events configuration parameter
num_iocleaners	num_iocleaners - Number of asynchronous page cleaners configuration parameter
num_ioservers	num_ioservers - Number of I/O servers configuration parameter
num_log_span	num_log_span - Number log span configuration parameter
num_quantiles	num_quantiles - Number of quantiles for columns configuration parameter
numarchretry	numarchretry - Number of retries on error configuration parameter
pckcachesz	pckcachesz - Package cache size configuration parameter
self_tuning_mem	self_tuning_mem- Self-tuning memory configuration parameter
sheapthres_shr	sheapthres_shr - Sort heap threshold for shared sorts configuration parameter
sortheap	sortheap - Sort heap size configuration parameter
stat_heap_sz	stat_heap_sz - Statistics heap size configuration parameter
stmt_conc	stmt_conc - Statement concentrator configuration parameter
stmtheap	stmtheap - Statement heap size configuration parameter
tsm_mgmtclass	tsm_mgmtclass - Tivoli Storage Manager management class configuration parameter
tsm_nodename	tsm_nodename - Tivoli Storage Manager node name configuration parameter
tsm_owner	tsm_owner - Tivoli Storage Manager owner name configuration parameter
tsm_password	tsm_password - Tivoli Storage Manager password configuration parameter
util_heap_sz	util_heap_sz - Utility heap size configuration parameter
wlm_collect_int	wlm_collect_int - Workload management collection interval configuration parameter

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
  ```

Restrictions and behavior when configuring max_coordagents and max_connections

The Version 9.5 default for the **max_coordagents** and **max_connections** parameters will be AUTOMATIC, with **max_coordagents** set to 200 and **max_connections** set to -1 (that is, set to the value of **max_coordagents**). These settings set Concentrator to OFF.

While configuring **max_coordagents** or **max_connections** online, there will be some restrictions and behavior to be aware of:

- If the value of **max_coordagents** is increased, the setting takes effect immediately and new requests will be allowed to create new coordinating agents. If the value is decreased, the number of coordinating agents will not be reduced immediately. Rather, the number of coordinating agents will no longer increase, and existing coordinating agents might terminate after finishing their current set of work, in order to reduce the overall number of coordinating agents. New requests for work that require a coordinating agent will not be serviced until the total number of coordinating agents falls below the new value and a coordinating agent becomes free.
- If the value for **max_connections** is increased, the setting takes effect immediately and new connections previously blocked because of this parameter will be allowed. If the value is decreased, the database manager will not actively terminate existing connections; instead, new connections will not be allowed until enough of the existing connections are terminated to bring the value down below the new maximum.
- If **max_connections** is set to -1 (default), then the maximum number of connections allowed is the same as **max_coordagents**, and when **max_coordagents** is updated offline or online; the maximum number of connections allowed will be updated as well.

While changing the value of **max_coordagents** or **max_connections** online, you cannot change it such that connection Concentrator will be turned either ON, if it's off, or OFF, if it's ON. For example, if at **START DBM** time **max_coordagents** is less than **max_connections** (Concentrator is ON), then all updates done online to these

two parameters must maintain the relationship **max_coordagents** < **max_connections**. Similarly, if at **START DBM** time, **max_coordagents** is greater than or equal to **max_connections** (Concentrator is OFF), then all updates done online must maintain this relationship.

When you perform this type of update online, the database manager does not fail the operation, instead it defers the update. The warning SQL1362W message is returned, similar to any case when updating the database manager configuration parameters where **IMMEDIATE** is specified, but is not possible.

When setting **max_coordagents** or **max_connections** to AUTOMATIC, the following behavior can be expected:

- Both of these parameters can be configured with a starting value and an AUTOMATIC setting. For example, the following command associates a value of 200 and AUTOMATIC to the **max_coordagents** parameter:

  ```
  UPDATE DBM CONFIG USING max_coordagents 200 AUTOMATIC
  ```

  These parameters will always have a value associated with them, either the value set as default, or some value that you specified. If only AUTOMATIC is specified when updating either parameter, that is, no value is specified, and the parameter previously had a value associated with it, that value would remain. Only the AUTOMATIC setting would be affected.

  Note: When Concentrator is ON, the values assigned to these two configuration parameters are important even when the parameters are set to AUTOMATIC.
- If both parameters are set to AUTOMATIC, the database manager allows the number of connections and coordinating agents to increase as needed to suit the workload. However, the following caveats apply:
  1. When Concentrator is OFF, the database manager maintains a one-to-one ratio: for every connection there will be only *one* coordinating agent.
  2. When Concentrator is ON, the database manager tries to maintain the ratio of coordinating agents to connections set by the values in the parameters.

     Note:

     – The approach used to maintain the ratio is designed to be unintrusive and does not guarantee the ratio will be maintained perfectly. New connections are always allowed in this scenario, though they may have to wait for an available coordinating agent. New coordinating agents will be created as needed to maintain the ratio. As connections are terminated, the database manager might also terminate coordinating agents to maintain the ratio

     – The database manager will not reduce the ratio that you set. The initial values of **max_coordagents** and **max_connections** that you set are considered a lower bound.
- The current and delayed values of both these parameters can be displayed through various means, such as CLP or APIs. The values displayed will always be the values set by the user. For example, if the following command were issued, and then 30 concurrent connections performing work on the instance were started, the displayed values for **max_connections** and **max_coordagents** will still be 20, AUTOMATIC:

  ```
  UPDATE DBM CFG USING max_connections 20 AUTOMATIC,
    max_coordagents 20 AUTOMATIC
  ```

To determine the real number of connections and coordinating agents currently running monitor elements, you can also use the Health Monitor.

- If **max_connections** is set to AUTOMATIC with a value greater than **max_coordagents** (so that Concentrator is ON), and **max_coordagents** is not set to AUTOMATIC, then the database manager allows an unlimited number of connections that will use only a limited number of coordinating agents.

  Note: Connections might have to wait for available coordinating agents.

The use of the AUTOMATIC option for the **max_coordagents** and **max_connections** configuration parameters is only valid in the following two scenarios:

1. Both parameters are set to AUTOMATIC
2. Concentrator is enabled with **max_connections** set to AUTOMATIC, while **max_coordagents** is not.

All other configurations using AUTOMATIC for these parameters will be blocked and will return SQL6112N, with a reason code that explains the valid settings of AUTOMATIC for these two parameters.

Database Manager configuration parameters

agent_stack_sz - Agent stack size

This parameter determines the memory that is allocated by DB2 for each agent thread stack.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]

Linux (32-bit)
256 [16 – 1024]

Linux (64-bit) and UNIX
1024 [256 – 32768]

Windows
16 [8 – 1000]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
Linux and UNIX: Stack space (process virtual memory) is allocated as needed or reused in the main DB2 server process when a thread is created. Stack memory is used or committed as necessary.

Windows: AGENT_STACK_SZ represents the initial committed stack memory when a thread is created. Additional stack memory is used or committed as necessary.

When freed

Linux and UNIX: Stack space (process virtual memory) is retained for reuse when threads terminate and are freed when the DB2 server shuts down.

Windows: Stack space and memory are freed when a thread terminates.

On UNIX and Linux, **agent_stack_sz** is rounded up to the next larger power-of-2 based value. The default settings should be sufficient for most workloads on Linux and UNIX systems.

In Windows environments, this parameter is used to set the initial committed stack size for each agent. Regardless of the setting, each agent stack can grow to the minimum reserved stack size of 256 KB on 32-bit versions of Windows, and 2 MB on 64-bit versions of Windows. Above this size, the agent stack may run out of space and return an error.

Windows uses the concepts of a "reserved" stack - the maximum to which the stack can grow, and the "committed" stack - the amount of memory committed to the stack when it is created. In addition, a guard page is added to the specified committed stack size in order to determine the minimum reserved stack space required. For example, with **agent_stack_sz** (committed stack) set to a value of 16, 1 guard page is added, and so the reserved stack size must be at least 17 pages. The maximum (that is, reserved) agent stack size can be increased by setting **agent_stack_sz** (that is, the committed stack) to a value that results in a minimum reserved stack size larger than the default reserve stack size of 64 pages. Note that Windows uses multiples of 1 MB for setting reserved stack sizes above 256 KB. For example, on 32-bit Windows, setting the agent stack size to a value within the range of [64 - 255] 4-KB pages results in a maximum stack of 1 MB, as 64*4 = 244 KB and 255*4 = 1020 KB round up to 1 MB. Setting the value for **agent_stack_sz** to a value less than the default reserve stack size has no effect on the maximum limit because the stack still grows as necessary up to the default reserve stack size.

You can change the default reserve stack size by using the **db2hdr** utility to change the header information for the `db2syscs.exe` file. The advantage of changing the default reserved stack size using the db2hdr utility is that it provides a finer granularity, therefore allowing the stack size to be set at the minimum required stack size. This conserves virtual address space on 32-bit Windows. However, you have to stop and restart DB2 for a change to the `db2syscs.exe` process to take effect, and this method must be repeated with any Fix Pack upgrade.

Recommendations:

If you are working with large or complex XML data in a 32-bit Windows environment, you should update the value of **agent_stack_sz** to at least 64 4-KB pages. Very complex XML schemas might require the value of **agent_stack_sz** to be much higher during schema registration or during XML document validation.

This limit is sufficient for most database operations.

Notes

- Agent stack memory does not count towards instance memory usage.

- While **agent_stack_sz** can be configured with a high stack space allocation for maximum usage, on average, only a small amount of allocated stack space is used by a thread. It is only this smaller amount which requires system memory.
- On the HP-UX platform, thread stack space requires a swap reservation. The approximate total swap requirement will be equal to the peak number of threads * (**agent_stack_sz**). The value of **agent_stack_sz** in this calculation is rounded up to the next larger power-of-2 value.

agentpri - Priority of agents

This parameter controls the priority given both to all agents, and to other database manager instance processes and threads, by the operating system scheduler. This priority determines how CPU time is given to the database manager processes, agents, and threads relative to the other processes and threads running on the machine.

Important: The **agentpri** database manager configuration is deprecated since Version 9.5. It can still be used in pre-Version 9.5 data servers and clients. Also, agent priority for the WLM service class has been deprecated in Version 10.1 and might be removed in a future release. Start to use the WLM dispatcher capability instead of agent priority. For more information, see "Agent priority of service classes has been deprecated" in *What's New for DB2 Version 10.1*.

Note: The following information applies only to pre-Version 9.5 data servers and clients.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]

AIX -1 (system) [41 - 125]

Other UNIX
-1 (system) [41 - 128]

Windows
-1 (system) [0 - 6]

Solaris
-1 (system) [0 - 59]

Linux -1 (system) [1 - 99]

HP-UX
-1 (system) [0 - 31]

When the parameter is set to -1 or system, no special action is taken and the database manager is scheduled in the normal way that the operating system schedules all processes and threads. When the parameter is set to a value other than -1 or system, the database manager will create its processes and threads with a static priority set to the value of the parameter. Therefore, this parameter allows

you to control the priority with which the database manager processes and threads (in a partitioned database environment, this also includes coordinating and subagents, the parallel system controllers, and the FCM daemons) will execute on your machine.

You can use this parameter to increase database manager throughput. The values for setting this parameter are dependent on the operating system on which the database manager is running. For example, in a Linux or UNIX environment, numerically low values yield high priorities. When the parameter is set to a value between 41 and 125, the database manager creates its agents with a UNIX static priority set to the value of the parameter. This is important in Linux and UNIX environments because numerically low values yield high priorities for the database manager, but other processes (including applications and users) might experience delays because they cannot obtain enough CPU time. You should balance the setting of this parameter with the other activity expected on the machine.

Restrictions:

- If you set this parameter to a non-default value on Linux and UNIX platforms, you cannot use the governor to alter agent priorities.
- On the Solaris operating system, you should not change the default value (-1). Changing the default value sets the priority of DB2 processes to real-time, which can monopolize all available resources on the system.

Recommendation: The default value should be used initially. This value provides a good compromise between response time to other users/applications and database manager throughput.

If database performance is a concern, you can use benchmarking techniques to determine the optimum setting for this parameter. You should take care when increasing the priority of the database manager because performance of other user processes can be severely degraded, especially when the CPU utilization is very high. Increasing the priority of the database manager processes and threads can have significant performance benefits.

alt_diagpath - Alternate diagnostic data directory path

This parameter allows you to specify the fully qualified alternate path for DB2 diagnostic information that is used when the primary diagnostic data path, **diagpath**, is unavailable.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Propagation class
Immediate

Default [range]
Null [any valid path name, , '"*pathname* $h"', '"*pathname*

$h/*trailing-dir*"', , '"*pathname* $n"',[2] '"*pathname* $n/*trailing-dir*"', '"*pathname* $m"', '"*pathname* $m/*trailing-dir*"', '"*pathname* hn"',[3] '"*pathname* hn/*trailing-dir*"', '"*pathname* hm"', or '"*pathname* hm/*trailing-dir*"']

Symbols

pathname
A directory path to use when the primary diagnostic data directory path is unavailable

$h Resolves to HOST_*hostname*

Note: Starting in Version 10, in DB2 pureScale environments, $h refers to the member's home host.

$n Resolves to NODE*number*

$m Resolves to DIAG_*number*. Note that DIAG_*number* is used regardless of whether it refers to a database partition, a CF, or a member.

/trailing-dir
A single directory, or a directory and sub-directory to trail $h or $n

The following values are available:

- '"*pathname* $h"'
- '"*pathname* $h/*trailing-dir*"'
- '"*pathname* $n"'
- '"*pathname* $n/*trailing-dir*"'
- '"*pathname* $m"'
- '"*pathname* $m/*trailing-dir*"'
- '"*pathname* hn"'
- '"*pathname* hn/*trailing-dir*"'
- '"*pathname* hm"'
- '"*pathname* hm/*trailing-dir*"'

The alternate diagnostic data directory can contain the same diagnostic data as the primary diagnostic data directory set with the **diagpath** parameter. When **alt_diagpath** is set and the primary diagnostic data directory becomes unavailable, diagnostic logging continues in the alternate diagnostic data directory path specified, then resumes in its original location when the primary diagnostic path becomes available again. If this parameter is null and the primary diagnostic data directory specified by the **diagpath** parameter is unavailable, no further diagnostic information is written until the primary diagnostic path becomes available again. For improved resilience, set the alternate diagnostic data directory to point to a different file system than the primary diagnostic data directory.

Starting in Version 10.1, the alternate diagnostic data directory path writes to private db2diag.log for each member and CF, by default. To revert to the behavior of previous releases, in which the diagnostic data is written to the same directory, specify the **alt_diagpath** with a *pathname* and no token ($h, $n, or $m).

Note:

2. $n is deprecated and might be removed in a future release.

3. hn is deprecated and might be removed in a future release.

- To avoid the operating system shell interpreting the $ sign on some Linux and UNIX systems, a single quote must be placed outside of the double quote, as shown in the syntax.
- In the CLP interactive mode, or if the command is read and executed from an input file, the double quote is not required.
- `$h`, `$m`, and `$n` are case insensitive.
- The dynamic behavior for **alt_diagpath** does not extend to all processes.
- The **db2sysc** DB2 server process can detect dynamic changes, for example, when you issue the **UPDATE DATABASE MANAGER CONFIGURATION** command over an instance attachment.
- When DB2 client and application processes start, they use the **alt_diagpath** configuration parameter setting and do not detect any dynamic changes.
- On UNIX systems, if both **diagpath** and **alt_diagpath** are not available, the db2 diagnostic message is dumped to the syslog file.
- There is no default directory for **alt_diagpath** configuration parameter.
- The **alt_diagpath** and **diagpath** configuration parameters are exclusive to each other. They cannot be set to same directory path.
- If **alt_diagpath** (or **diagpath**) is unavailable that means diagnostic data dumping failed due to an error, such as: The directory was deleted, a disk error, disk is lost, network problems, file permission error, or disk is full.

alternate_auth_enc - Alternate encryption algorithm for incoming connections at server configuration parameter

This configuration parameter specifies the alternate encryption algorithm used to encrypt the user IDs and passwords submitted to a DB2 database server for authentication. Specifically, this parameter affects the encryption algorithm when the authentication method negotiated between the DB2 client and the DB2 database server is SERVER_ENCRYPT.

Configuration type
Database manager

Applies to
- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
`NOT_SPECIFIED` [`AES_CMP`; `AES_ONLY`]

The user ID and password submitted for authentication on the DB2 database server are encrypted when the authentication method negotiated between the DB2 client and the DB2 server is SERVER_ENCRYPT. The authentication method negotiated depends on the authentication type setting on the server and the authentication type requested by the client. 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. It can be either DES or AES depending on this setting.

When the default (`NOT_SPECIFIED`) value is used, the database server accepts the encryption algorithm that the client proposes.

When **alternate_auth_enc** is set to AES_ONLY, the database server will only accept connections that use AES encryption. If the client does not support AES encryption, then the connection is rejected.

When **alternate_auth_enc** is set to AES_CMP, the database server will accept user IDs and passwords that are encrypted using either AES or DES, but it will negotiate for AES if the client supports AES encryption.

aslheapsz - Application support layer heap size

The application support layer heap represents a communication buffer between the local application and its associated agent. This buffer is allocated as shared memory by each database manager agent that is started.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
15 [1 - 524 288]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
When the database manager agent process is started for the local application

When freed
When the database manager agent process is terminated

If the request to the database manager, or its associated reply, do not fit into the buffer they will be split into two or more send-and-receive pairs. The size of this buffer should be set to handle the majority of requests using a single send-and-receive pair. The size of the request is based on the storage required to hold:

- The input SQLDA
- All of the associated data in the SQLVARs
- The output SQLDA
- Other fields which do not generally exceed 250 bytes.

In addition to this communication buffer, this parameter is also used for two other purposes:

- It is used to determine the I/O block size when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the Data Server Runtime Client

cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

- It is used to determine the communication size between agents and **db2fmp** processes. (A **db2fmp** process can be a user-defined function or a fenced stored procedure.) The number of bytes is allocated from shared memory for each **db2fmp** process or thread that is active on the system.

The data sent from the local application is received by the database manager into a set of contiguous memory allocated from the query heap. The **aslheapsz** parameter is used to determine the initial size of the query heap (for both local and remote clients). The maximum size of the query heap is defined by the **query_heap_sz** parameter.

Recommendation: If your application's requests are generally small and the application is running on a memory constrained system, you might want to reduce the value of this parameter. If your queries are generally very large, requiring more than one send and receive request, and your system is not constrained by memory, you might want to increase the value of this parameter.

Use the following formula to calculate a minimum number of pages for **aslheapsz**:

```
aslheapsz >= ( sizeof(input SQLDA)
             + sizeof(each input SQLVAR)
             + sizeof(output SQLDA)
             + 250 ) / 4096
```

where sizeof(x) is the size of x in bytes that calculates the number of pages of a given input or output value.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks might yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks might also cause more data than required to be block fetched by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application.

audit_buf_sz - Audit buffer size

This parameter specifies the size of the buffer used when auditing the database.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
0 [0 - 65 000]

Unit of measure
Pages (4 KB)

When allocated
When DB2 is started

When freed
When DB2 is stopped

The default value for this parameter is zero (0). If the value is zero (0), the audit buffer is not used. If the value is greater than zero (0), space is allocated for the audit buffer where the audit records will be placed when they are generated by the audit facility. The value times 4 KB pages is the amount of space allocated for the audit buffer. At regular time intervals or when the audit buffer is full, the **db2auditd** audit daemon process flushes the audit buffer to disk. The audit buffer cannot be allocated dynamically; DB2 must be stopped and then restarted before the new value for this parameter takes effect.

By changing this parameter from the default to some value larger than zero (0), the audit facility writes records to disk asynchronously compared to the execution of the statements generating the audit records. This improves DB2 performance over leaving the parameter value at zero (0). The value of zero (0) means the audit facility writes records to disk synchronously with (at the same time as) the execution of the statements generating the audit records. The synchronous operation during auditing decreases the performance of applications running in DB2.

authentication - Authentication type

This parameter specifies and determines how and where authentication of a user takes place.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
SERVER [CLIENT; SERVER; SERVER_ENCRYPT; DATA_ENCRYPT; DATA_ENCRYPT_CMP; KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

If **authentication** is SERVER, the user ID and password are sent from the client to the server so that authentication can take place on the server. The value SERVER_ENCRYPT provides the same behavior as SERVER, except that any user IDs and passwords sent over the network are encrypted.

A value of DATA_ENCRYPT means the server accepts encrypted SERVER authentication schemes and the encryption of user data. The authentication works exactly the same way as SERVER_ENCRYPT.

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 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) streaming
- SQLDA descriptors

A value of DATA_ENCRYPT_CMP means the server accepts encrypted SERVER authentication schemes and the encryption of user data. In addition, this authentication type allows compatibility with earlier products that do not support 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.

Note: For a standards compliance (defined in the "Standards compliance" topic) configuration, SERVER is the only supported value.

A value of CLIENT indicates that all authentication takes place at the client. No authentication needs to be performed at the server.

A value of KERBEROS means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication. With an authentication type of KRB_SERVER_ENCRYPT at the server and clients that support the Kerberos security system, the effective system authentication type is KERBEROS. If the clients do not support the Kerberos security system, the system authentication type is effectively equivalent to SERVER_ENCRYPT.

A value of GSSPLUGIN means that authentication is performed using an external GSSAPI-based security mechanism. With an authentication type of GSS_SERVER_ENCRYPT at the server and clients that support the GSSPLUGIN security mechanism, the effective system authentication type is GSSPLUGIN (that is, if the clients support one of the server's plug-ins). If the clients do not support the GSSPLUGIN security mechanism, the system authentication type is effectively equivalent to SERVER_ENCRYPT.

Recommendation: Typically, the default value (SERVER) is adequate for local clients. If remote clients are connecting to the database server then SERVER_ENCRYPT is the suggested value to protect the user ID and password.

cf_diaglevel - diagnostic error capture level configuration parameter for the CF

This parameter specifies the type of diagnostic errors that will be recorded in the cfdiag*.log files

Configuration type
Database Manager

Applies to
- DB2 pureScale

Parameter type
Configurable offline

Propagation class
Immediate

Default [range]
2 [1 - 4]

Valid values for this parameter are:

- 0 - No diagnostic data captured
- 1 - Severe errors only
- 2 - All errors
- 3 - All errors and warnings
- 4 - All errors, warnings and informational messages

The **cf_diagpath** configuration parameter is used to specify the directory that will contain the diagnostic message file that will be generated, based on the value of the **cf_diaglevel** parameter

The CF log file location will be determined in the following order:

- **cf_diagpath** (DBM configuration parameter)
- **diagpath** (DBM configuration parameter)
- **DB2PATH**/db2dump

cf_diagpath - diagnostic data directory path configuration parameter for the CF

This parameter allows you to specify the fully qualified path for the diagnostic information file for the CF

Configuration type
Database Manager

Applies to

- DB2 pureScale

Parameter type
Configurable offline

Propagation class
Immediate

Default [range]
INSTHOME/sqllib/db2dump/ $m [any valid path name]

Starting in Version 10.1, the cf diagnostic data directory path writes to a private db2diag.log for each CF by default. To revert to the behavior of previous releases, in which the diagnostic data for the CF is written to the same directory, specify the **cf_diagpath** with a *pathname* and no token.

Each CF log file name has the following format:

```
cfdiag-YYYYMMDDhhmmssuuuuuu.<cf#>.log
```

However, there is also a single static CF diagnostic log name that always points to the most current CF diagnostic logging file and has the following format:

```
cfdiag.<cf#>.log
```

For example, if you listed all of the CF log files, you would see something similar to the following:

```
$ ls -la cfdiag*
lrwxrwxrwx 1 db2inst1 pdxdb2   35 2011-02-09 15:07
      cfdiag.128.log -> cfdiag-20110209150733000049.128.log
```

```
lrwxrwxrwx 1 db2inst1 pdxdb2   35 2011-02-09 15:10
      cfdiag.129.log -> cfdiag-20110209151021000040.129.log
-rw-r-xr-- 1 db2inst1 pdxdb2 1271 2011-02-09 15:07
      cfdiag-20110209150733000049.128.log
-rw-r-xr-- 1 db2inst1 pdxdb2 1271 2011-02-09 15:07
      cfdiag-20110209150740000082.129.log
-rw-r-xr-- 1 db2inst1 pdxdb2 1274 2011-02-09 15:10
      cfdiag-20110209151021000040.129.log
```

In this example, `cfdiag.128.log` and `cfdiag.129.log` are symbolic links to the latest versions of the otherwise time stamped log files.

The CF log file looks similar to the `db2diag.log file`.

cf_mem_sz - CF memory configuration parameter

This parameter controls the total memory that is used by the cluster caching facility, also known as CF.

Configuration type
Database Manager

Applies to
Applies to a DB2 pureScale instance only.

- Database server with local and remote clients

Parameter type
Configurable offline

Default [range]
AUTOMATIC [32768 - 4 294 967 295]

Unit of measure
Pages (4 KB)

When allocated
When the CF is started

When freed
When the CF is stopped

When the default setting AUTOMATIC is applied, the amount of cluster caching facility memory is determined by querying the total memory that is available on the CF server. The parameter is then set to either an appropriate percentage of the total memory on the CF server, or the amount of free memory on the machine, whichever value is less. An appropriate percentage is typically 70% to 90% of the total memory available on the CF. Factors for consideration during AUTOMATIC computation also include:

- If the CF and any DB2 members coexist
- If there are multiple instances on the same host

cf_num_conns - Number of CF connections per member per CF configuration parameter

This parameter controls the initial size of the cluster caching facility (CF) connection pool.

Configuration type
Database Manager

Parameter type
Configurable online

Default [range]
AUTOMATIC [4 - 256]

When allocated
When DB2 is started

When freed
When DB2 is stopped

When you set the **cf_num_conns** parameter to AUTOMATIC (the default), the DB2 database manager creates an initial number of CF connections for each member with each CF at start time. This initial number is based on the number of worker threads, (See: "cf_num_workers - Number of worker threads configuration parameter"), number of connections per worker thread, and the number of members in the cluster. The actual maximum value for the **cf_num_conns** parameter is calculated automatically by DB2 at member startup, based on those parameters.

If the **cf_num_conns** parameter is set to AUTOMATIC and then raised to a value higher than the current value, with an instance attachment, new CF connections are created. However, if you set the **cf_num_conns** parameter to a value lower than the current value, CF connections are not closed.

When the **cf_num_conns** parameter is set to AUTOMATIC, any dropped connections caused by a port failure (or other network problem) are reestablished evenly on the remaining ports. When a failed port comes back online, connections are rebalanced as required.

When you set the **cf_num_conns** parameter to a fixed numeric value, the DB2 database manager creates exactly that number of CF connections for each member with each CF at start time. There is no automatic growing or shrinking done by DB2 database manager.

If the **cf_num_conns** parameter is set to a fixed value, and then the value is increased or decreased, with an instance attachment, then the number of CF connections is increased or decreased immediately, according to the new value.

When the **cf_num_conns** parameter is set to a fixed value, a port failure does not increase the number of connections on the remaining ports. The number of connections made to each port remains at the number found by dividing the **cf_num_conns** parameter by the number of ports. That number does not change, even if some ports are offline.

cf_num_workers - Number of worker threads configuration parameter

The **cf_num_workers** parameter specifies the total number of worker threads on the cluster caching facility (CF). Worker threads are distributed among the communication adapter ports to balance the number of worker threads servicing requests on each interface.

Configuration type
Database Manager

Applies to
Applies to a DB2 pureScale instance only.

Parameter type
Configurable offline

Default [range]
AUTOMATIC [1 - 31]

When set to the default (AUTOMATIC), the parameter value is configured to be one less than the number of available processors on the CF. The available CPUs are equally divided among the instances before one processor is subtracted from the resulting value for each instance. If there are coexisting members on the CF host, the number of worker threads on the CF is further divided by the total number of CF and members on a host.

If there is only one processor on the CF server machine, this value is set to 1.

The total number of CF worker threads is shown in the cfdiag.log file.

To calculate the number of worker threads assigned to each cluster interconnect interface, divide the total number of CF work threads by the number of communication adapter ports defined for the CF.

```
Number of workers assigned to each interface =
    (cf_num_workers) / (number of interfaces)
```

When the **cf_num_workers** parameter is set to AUTOMATIC, the database manager configures the number of worker threads so that it is equally divisible by the number of communication adapter ports configured for the CF.

For example, on a CF server machine with 7 available processors, and 2 communication adapter ports, the value for the AUTOMATIC setting is:

```
Total CF worker threads =
    (7 processors) - (1 processor to prevent using all processors) = 6
Thus, number of workers connected to each cluster interconnect interface =
    6 / 2 = 3
```

If you set the **cf_num_workers** parameter manually, set the value to be equal or greater than the number of communication adapter ports so that there is at least one worker thread for each interface. If there is an insufficient number of worker threads to cover all interfaces, an alert is logged for the CF, which will fail to start. The parameter value must be changed to address the problem.

Restrictions:
If you are running InfiniBand or uDAPL, do not set this value higher than the number of processors on the CF server machine. Each worker thread operates on a processor, waiting for uDAPL communication. Performance is affected if the number of worker threads exceeds the number of processors.

catalog_noauth - Cataloging allowed without authority

This parameter specifies whether users are able to catalog and uncatalog databases and nodes, or DCS and ODBC directories, without SYSADM authority.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]

Database server with local and remote clients
NO [NO (0) - YES (1)]

Client; Database server with local clients
YES [NO (0) - YES (1)]

The default value (0) for this parameter indicates that SYSADM authority is required. When this parameter is set to 1 (yes), SYSADM authority is not required.

clnt_krb_plugin - Client Kerberos plug-in

This parameter specifies the name of the default Kerberos plug-in library to be used for client-side authentication and local authorization.

Configuration type
Database manager

Applies to

- Client
- Database server with local and remote clients
- Database server with local clients only
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null on Linux and UNIX operating systems and `IBMkrb5` on Windows operating systems [any valid string]

The plug-in is used when the client is authenticated using Kerberos authentication or when local authorization is performed and the authentication type in the database manager configuration is set to `KERBEROS`.

clnt_pw_plugin - Client userid-password plug-in

This parameter specifies the name of the userid-password plug-in library to be used for client-side authentication and local authorization.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [any valid string]

By default, the value is null and the DB2 supplied userid-password plug-in library is used. The plug-in is used when the client is authenticated using CLIENT authentication, or when local authorization is performed and the **authentication** in the DBM CFG is CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP. For non-root installations, if the DB2 userid and password plug-in library is used, the **db2rfe** command must be run before using your DB2 database product.

cluster_mgr - Cluster manager name

This parameter enables the database manager to communicate incremental cluster configuration changes to the specified cluster manager.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Multi-partitioned database server with local and remote clients

Parameter type
Informational

Default

- In a DB2 pureScale environment: TSA; otherwise, no default

Valid values

- TSA

This parameter is set automatically when you are installing the DB2 pureScale Feature or if you are using the DB2 High Availability Instance Configuration Utility (**db2haicu**) to configure your cluster for high availability.

comm_bandwidth - Communications bandwidth

This parameter helps the query optimizer determine access paths by indicating the bandwidth between database partition servers.

Configuration type
Database manager

Applies to
Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Statement boundary

Default [range]
-1 [-1, 0.1 - 100000]

A value of -1 causes the parameter value to be reset to the default. The default value is calculated based on the speed of the underlying communications adapter. A value of 100 can be expected for systems using Gigabit Ethernet.

Unit of measure
Megabytes per second

The value calculated for the communications bandwidth, in megabytes per second, is used by the query optimizer to estimate the cost of performing certain operations between the database partition servers of a partitioned database system. The optimizer does not model the cost of communications between a client and a server, so this parameter should reflect only the nominal bandwidth between the database partition servers, if any.

You can explicitly set this value to model a production environment on your test system or to assess the impact of upgrading hardware.

Recommendation: You should only adjust this parameter if you want to model a different environment.

The communications bandwidth is used by the optimizer in determining access paths. You should consider rebinding applications (using the **REBIND PACKAGE** command) after changing this parameter.

comm_exit_list - Communication buffer exit library list

This parameter specifies the list of communication buffer exit libraries that DB2 will use. A communication buffer exit library is a dynamically loaded library that vendor applications can use to gain access to and examine DB2 communication buffers used to communicate with client applications.

Configuration type
Database Manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
`Null [any valid string]`

By default, the value is null. When the value is non-null, it is taken to be a comma separate list of communication buffer exit libraries that DB2 is to use. Each library must be separated by a comma, with no spaces either before or after the comma.

Each name can be up to 32 bytes long. The name should not include the extension (eg. .so or .a). The total length of the parameter must not be more than 128 bytes.

conn_elapse - Connection elapse time

This parameter specifies the number of seconds within which a network connection is to be established between DB2 members.

Configuration type
Database manager

Applies to
DB2 pureScale server (with more than one DB2 member)

Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
10 [0-100]

Unit of measure
Seconds

If the attempt to connect succeeds within the time specified by this parameter, communications are established. If it fails, another attempt is made to establish communications. If the connection is attempted the number of times specified by the **max_connretries** parameter and always times out, an error is issued.

cpuspeed - CPU speed

This parameter reflects the CPU speed of the machine(s) the database is installed on.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Propagation class
Statement boundary

Default [range]
-1 [$1x10^{-10}$ - 1] A value of -1 will cause the parameter value to be reset based on the running of the measurement program.

Unit of measure
Milliseconds

This program is executed if benchmark results are not available if the data for the IBM RS/6000® model 530H is not found in the file, or if the data for your machine is not found in the file.

You can explicitly set this value to model a production environment on your test system or to assess the impact of upgrading hardware. By setting it to -1, **cpuspeed** will be re-computed.

Recommendation: You should only adjust this parameter if you want to model a different environment.

The CPU speed is used by the optimizer in determining access paths. You should consider rebinding applications (using the **REBIND PACKAGE** command) after changing this parameter.

cur_commit - Currently committed configuration parameter

This parameter controls the behavior of cursor stability (CS) scans.

Configuration type
Database

Parameter type

- Configurable
- Configurable by member in a DB2 pureScale environment

Default [range]

ON [ON, AVAILABLE, DISABLED]

For new databases, the default is set to ON. When the default is set to ON your query will return the currently committed value of the data at the time when your query is submitted.

During database upgrade from V9.5 or earlier, the **cur_commit** configuration parameter is set to DISABLED to maintain the same behavior as in previous releases. If you want to use currently committed on cursor stability scans, you need to set the **cur_commit** configuration parameter to ON after the upgrade.

You can explicitly set the **cur_commit** configuration parameter to AVAILABLE. Once you set this parameter, you need to explicitly request for currently committed behavior to see the results that are currently committed.

Note: Three registry variables **DB2_EVALUNCOMMITTED**, **DB2_SKIPDELETED**, and **DB2_SKIPINSERTED** are affected by currently committed when cursor stability isolation level is used. These registry variables are ignored when **USE CURRENTLY COMMITTED** or **WAIT FOR OUTCOME** are specified explicitly on the **BIND** or at statement prepare time.

Note: Performance considerations may be applicable in a database where there are significant lock conflicts when using currently committed. The committed version of the row is retrieved from the log, and will perform better and avoid log disk activity when the log record is still in the log buffer. Therefore, to improve the performance of retrieving previously committed data, you might consider an increase to the value of the **logbufsz** parameter.

date_compat - Date compatibility database configuration parameter

This parameter indicates whether the DATE compatibility semantics associated with the TIMESTAMP(0) data type are applied to the connected database.

Configuration type

Database

Parameter type

Informational

The value is determined at database creation time, and is based on the setting of the **DB2_COMPATIBILITY_VECTOR** registry variable for DATE data type. The value cannot be changed.

dft_account_str - Default charge-back account

This parameter acts as the default suffix of accounting identifiers.

Configuration type

Database manager

Applies to

- Database server with local and remote clients

- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
Null [any valid string]

With each application connect request, an accounting identifier consisting of a DB2 Connect-generated prefix and the user supplied suffix is sent from the application requester to a DRDA application server. This accounting information provides a mechanism for system administrators to associate resource usage with each user access.

Note: This parameter is only applicable to DB2 Connect.

The suffix is supplied by the application program calling the sqlesact() API or the user setting the environment variable **DB2ACCOUNT**. If a suffix is not supplied by either the API or environment variable, DB2 Connect uses the value of this parameter as the default suffix value. This parameter is particularly useful for earlier database clients (anything before version 2) that do not have the capability to forward an accounting string to DB2 Connect.

Recommendation: Set this accounting string using the following possible values:
- Alphabetics (A through Z)
- Numerics (0 through 9)
- Underscore (_).

dft_monswitches - Default database system monitor switches

This parameter allows you to set a number of switches which are each internally represented by a bit of the parameter.

Configuration type
Database manager

Applies to
- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Note: The change takes effect immediately if you explicitly ATTACH to the instance before modifying the dft_mon_xxxx switch settings. Otherwise the setting takes effect the next time the instance is restarted.

Default
All switches turned off, except dft_mon_timestamp, which is turned on by default

The parameter is unique in that you can update each of these switches independently by setting the following parameters:

dft_mon_uow
Default value of the snapshot monitor's unit of work (UOW) switch

dft_mon_stmt
Default value of the snapshot monitor's statement switch

dft_mon_table
Default value of the snapshot monitor's table switch

dft_mon_bufpool
Default value of the snapshot monitor's buffer pool switch

dft_mon_lock
Default value of the snapshot monitor's lock switch

dft_mon_sort
Default value of the snapshot monitor's sort switch

dft_mon_timestamp
Default value of the snapshot monitor's timestamp switch

Recommendation: Any switch (except dft_mon_timestamp) that is turned ON instructs the database manager to collect monitor data related to that switch. Collecting additional monitor data increases the processing time of the database manager which can impact system performance. Turning the dft_mon_timestamp switch OFF becomes important as CPU utilization approaches 100%. When this occurs, the CPU time required for issuing timestamps increases dramatically. Furthermore, if the timestamp switch is turned OFF, the overall cost of other data under monitor switch control is greatly reduced.

All monitoring applications inherit these default switch settings when the application issues its first monitoring request (for example, setting a switch, activating the event monitor, taking a snapshot). You should turn on a switch in the configuration file only if you want to collect data starting from the moment the database manager is started. (Otherwise, each monitoring application can set its own switches and the data it collects becomes relative to the time its switches are set.)

dftdbpath - Default database path

This parameter contains the default file path used to create databases under the database manager. If no path is specified when a database is created, the database is created under the path specified by the **dftdbpath** parameter.

Configuration type
Database manager

Applies to
- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]

UNIX Home directory of instance owner [any existing path]

Windows
Drive on which the DB2 database system is installed [any existing path]

In a partitioned database environment, you should ensure that the path on which the database is being created is not an NFS-mounted path (on Linix and UNIX operating systems), or a network drive (in a Windows environment). The specified path must physically exist on each database partition server. To avoid confusion, it is best to specify a path that is locally mounted on each database partition server. The maximum length of the path is 205 characters. The system appends the database partition name to the end of the path.

Given that databases can grow to a large size and that many users could be creating databases (depending on your environment and intentions), it is often convenient to be able to have all databases created and stored in a specified location. It is also good to be able to isolate databases from other applications and data both for integrity reasons and for ease of backup and recovery.

For Linux and UNIX environments, the length of the **dftdbpath** name cannot exceed 215 characters and must be a valid, absolute, path name. For Windows, the **dftdbpath** can be a drive letter, optionally followed by a colon.

Recommendation: If possible, put high volume databases on a different disk than other frequently accessed data, such as the operating system files and the database logs.

diaglevel - Diagnostic error capture level

This parameter specifies the type of diagnostic errors that will be recorded in the db2diag log file.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
3 [0 — 4]

Valid values for this parameter are:

- **0** – Starting in Fix Pack 1, only administration notification messages are captured on the client side. Critical errors, event messages, and administration notification messages are still captured on the server side.
- **1** – Only severe errors, critical errors, event messages, and administration notification messages are captured.
- **2** – All errors, event messages, and administration notification messages are captured.
- **3** – All errors, warnings, event messages, and administration notification messages are captured.
- **4** – All errors, warnings, informational messages, event messages, and administration notification messages are captured.

The **diagpath** configuration parameter is used to specify the directory that will contain the error file, alert log file, and any dump files that might be generated, based on the value of the **diaglevel** parameter.

Usage notes

- The dynamic behaviour for **diaglevel** does not extend to all processes.
- The **db2sysc** DB2 server process can detect dynamic changes, for example, when you issue the **UPDATE DATABASE MANAGER CONFIGURATION** command over an instance attachment.
- When DB2 client and application processes start, they use the **diaglevel** configuration parameter setting and do not detect any dynamic changes.
- To help resolve a problem, you can increase the value of this parameter to gather additional problem determination data.

diagpath - Diagnostic data directory path

This parameter allows you to specify the fully qualified primary path for DB2 diagnostic information.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Propagation class
Immediate

Default [range]
$*INSTHOME*/sqllib/db2dump/ $m [any valid path name (see the following section for resolution variables options)]

Symbols

pathname
A directory path to use instead of the default diagnostic data directory

$h Resolves to HOST_*hostname*.

Note: Starting in Version 10, in DB2 pureScale environments, $h refers to the member's home host.

$n Resolves to NODE*number*

$m Resolves to DIAG*number*. Note that DIAG*number* is used regardless of whether it refers to a database partition, a CF, or a member.

/trailing-dir
A single directory, or a directory and sub-directory to trail $h or $n

The following values are available:

- '"$h"'
- '"$h/*trailing-dir*"'
- '"*pathname* $h"'
- '"*pathname* $h/*trailing-dir*"'
- '"$n"'[4]
- '"$n/*trailing-dir*"'
- '"*pathname* $n"'
- '"*pathname* $n/*trailing-dir*"'
- '"$m"'
- '"$m/*trailing-dir*"'
- '"*pathname* $m"'
- '"*pathname* $m/*trailing-dir*"'
- '"hn"'[5]
- '"hn/*trailing-dir*"'
- '"*pathname* hn"'
- '"*pathname* hn/*trailing-dir*"'
- '"hm"'
- '"hm/*trailing-dir*"'
- '"*pathname* hm"'
- '"*pathname* hm/*trailing-dir*"'

The primary diagnostic data directory could possibly contain dump files, trap files, an error log, a notification file, an alert log file, and first occurrence data collection (FODC) packages, depending on your platform.

If this parameter is null, the diagnostic information will be written to a default diagnostic path directory string in one of the following directories or folders:

- In Windows environments:
  - The default location of user data files, for example, files under instance directories, varies from edition to edition of the Windows family of operating systems. Use the **DB2SET DB2INSTPROF** command to get the location of the instance directory. The file is in the instance subdirectory of the directory specified by the **DB2INSTPROF** registry variable.
- In Linux and UNIX environments: Information is written to *INSTHOME*/sqllib/db2dump/ $m, where *INSTHOME* is the home directory of the instance.

4. $n is deprecated and might be removed in a future release.

5. hn is deprecated and might be removed in a future release.

Starting in Version 10.1, the diagnostic data directory path writes to a private db2diag log file for each member and CF, by default. To revert to the behavior of previous releases, in which the diagnostic data is written to the same directory, specify the **diagpath** with a *pathname* and no token ($h, $n, or $m).

To split the diagnostic data directory path to collect diagnostic information per physical host, set the parameter to one of the following values:

- Split default diagnostic data directory path:

  ```
  db2 update dbm cfg using diagpath '"$h"'
  ```

  which creates a subdirectory under the default diagnostic data directory with the host name, as in the following:

  Default_diagpath/HOST_*hostname*
- Split default diagnostic data directory path with a trailing directory:

  ```
  db2 update dbm cfg using diagpath '"$h/trailing-dir"'
  ```

  which creates a subdirectory under the default diagnostic data directory with the host name and a trailing directory, as in the following:

  Default_diagpath/HOST_*hostname*/*trailing-dir*
- Split your own specified diagnostic data directory path (there is a blank space between *pathname* and $h):

  ```
  db2 update dbm cfg using diagpath '"pathname $h"'
  ```

  which creates a subdirectory under your own specified diagnostic data directory with the host name, as in the following:

  pathname/HOST_*hostname*
- Split your own specified diagnostic data directory path (there is a blank space between *pathname* and $h) and with a trailing directory:

  ```
  db2 update dbm cfg using diagpath '"pathname $h/trailing-dir"'
  ```

  which creates a subdirectory under your own specified diagnostic data directory with the host name and a trailing directory, as in the following:

  pathname/HOST_*hostname*/*trailing-dir*

To split the diagnostic data directory path to collect diagnostic information per physical host and per database partition per physical host, set the parameter to one of the following values:

- Split default diagnostic data directory path:

  ```
  db2 update dbm cfg using diagpath '"$h$n"'
  ```

  which creates a subdirectory for each logical partition on the host under the default diagnostic data directory with the host name and the partition number, as in the following:

  Default_diagpath/HOST_*hostname*/NODE*number*
- Split default diagnostic data directory path with a trailing directory:

  ```
  db2 update dbm cfg using diagpath '"$h$n/trailing-dir"'
  ```

  which creates a subdirectory for each logical partition on the host under the default diagnostic data directory with the host name, the partition number, and a trailing directory, as in the following:

  Default_diagpath/HOST_*hostname*/NODE*number*/*trailing-dir*

- Split your own specified diagnostic data directory path (there is a blank space between *pathname* and hn):

  ```
  db2 update dbm cfg using diagpath '"pathname $h$n"'
  ```

  which creates a subdirectory for each logical partition on the host under your own specified diagnostic data directory with the host name and the partition number, as in the following:

  pathname/HOST_*hostname*/NODE*number*

  For example, an AIX host, named boson, has 3 database partitions with node numbers 0, 1, and 2. An example of a list output for the directory is similar to the following:

  ```
  usr1@boson /home/user1/db2dump->ls -R *
  HOST_boson:

  HOST_boson:
  NODE0000 NODE0001 NODE0002

  HOST_boson/NODE0000:
  db2diag.log db2eventlog.000 db2resync.log db2sampl_Import.msg events usr1.nfy

  HOST_boson/NODE0000/events:
  db2optstats.0.log

  HOST_boson/NODE0001:
  db2diag.log db2eventlog.001 db2resync.log usr1.nfy stmmlog

  HOST_boson/NODE0001/stmmlog:
  stmm.0.log

  HOST_boson/NODE0002:
  db2diag.log db2eventlog.002 db2resync.log usr1.nfy
  ```

- Split your own specified diagnostic data directory path (there is a blank space between *pathname* and hn) and with a trailing directory:

  ```
  db2 update dbm cfg using diagpath '"pathname $h$n/trailing-dir"'
  ```

  which creates a subdirectory for each logical partition on the host under your own specified diagnostic data directory with the host name, the partition number and a trailing directory, as in the following:

  pathname/HOST_*hostname*/NODE*number*/*trailing-dir*

Note:

- To avoid the operating system shell interpreting the $ sign on some Linux and UNIX systems, a single quote must be placed outside of the double quote, as shown in the syntax.
- In the CLP interactive mode, or if the command is read and executed from an input file, the double quote is not required.
- $h, $n, and $m are case insensitive.
- The dynamic behavior for **diagpath** does not extend to all processes
- The **db2sysc** DB2 server process can detect dynamic changes, for example, when you issue the **UPDATE DATABASE MANAGER CONFIGURATION** command over an instance attachment.
- When DB2 client and application processes start, they use the **diagpath** configuration parameter setting and do not detect any dynamic changes.

In Version 9.5 and later, the default value of **DB2INSTPROF** at the global level is stored at the new location shown previously. Other profile registry variables that

specify the location of the runtime data files should query the value of **DB2INSTPROF**. The other variables include the following ones:

- **DB2CLIINIPATH**
- **DIAGPATH**
- **SPM_LOG_PATH**

Note: In DB2 Version 9.7 Fix Pack 4 and later fix packs, diagnostic logging can be made more resilient by setting an alternate diagnostic path in conjunction with the **diagpath** parameter. When **alt_diagpath** is set and the path specified by **diagpath** becomes unavailable, diagnostic logging continues in the alternate diagnostic data directory path specified, then resumes when the primary diagnostic path becomes available again.

diagsize - Rotating diagnostic and administration notification logs configuration parameter

This parameter helps control the maximum sizes of the diagnostic log and administration notification log files.

Configuration type
Database manager

Parameter type
Not configurable online

Default
0

Minimum value for specifying the size of rotating logs:
2

Maximum value for specifying the size of rotating logs:
The amount of free space in the directory specified by diagpath

Unit of measure
Megabytes

If the value of this parameter is `0`, the default, there is only one diagnostic log file, called the `db2diag.log` file. There is also only one administration notification log file, called the `<instance>.nfy` file, which is used only on Linux and UNIX operating systems. The sizes of these files can increase indefinitely.

If you set the parameter to a non-zero value and restart the <instance>, a series of rotating diagnostic log files and a series of rotating administration notification log files are used. These files are called the `db2diag.n.log` and `<instance>.n.nfy` files, where *n* is an integer; `<instance>.n.nfy` files apply only to Linux and UNIX operating systems. The number of `db2diag.n.log` files and `<instance>.n.nfy` files cannot exceed 10 each. When the size of 10th file is full, the oldest file is deleted, and a new file is created.

For example, on Linux and UNIX operating systems the rotating log files under **diagpath** might look like the following output:

```
db2diag.14.log, db2diag.15.log, ... , db2diag.22.log, db2diag.23.log
<instance>.0.nfy, <instance>.1.nfy...,<instance>.8.nfy,<instance>.9.nfy
```

If db2diag.23.log is full, db2diag.14.log will be deleted, db2diag.24.log will be created for `db2diag` logging

If <instance>.9.nfy is full, <instance>.0.nfy is deleted, <instance>.10.nfy will be created for administration notification logging.

Note that the messages are always logged to rotating log file with the largest index number db2diag.*largest n*.log, <*instance*>.*largest n*.nfy

The total size of the db2diag.*n*.log and <*instance*>.*n*.nfy files are determined by the value of the **diagsize** configuration parameter. By default, except on Windows operating systems, 90% of the value of **diagsize** is allocated to the db2diag.*n*.log files, and 10% of the value of **diagsize** is allocated to the <*instance*>.*n*.nfy files. For example, if you set **diagsize** to 1024 on a Linux or UNIX operating system, the total size of the db2diag.*n*.log files cannot exceed 921.6 MB, and the total size of the <*instance*>.*n*.nfy files cannot exceed 102.4 MB. On Windows operating systems, the total value of **diagsize** is allocated to the db2diag.*n*.log files. The size of each log file is determined by the total amount of space allocated to each type of log file divided by 10. For example, if the total size of the db2diag.*n*.log files cannot exceed 921.6 MB, the size of each db2diag.*n*.log file is 92.16 MB.

The maximum value that you specify for the **diagsize** configuration parameter cannot exceed the amount of free space in the directory that you specify for the **diagpath** configuration parameter. The diagnostic and administration notification log files are stored in this directory. To avoid losing information too quickly because of file rotation (the deletion of the oldest log file), set **diagsize** to a value that is greater than 50 MB but not more than 80% of the free space in the directory that you specify for **diagpath**.

For example,

- To set the **diagsize** to 1024 MB, that will switch to rotate logging behavior when DB2 gets restarted, use the following command:

  ```
  db2 update dbm cfg using diagsize 1024
  ```

- To set the **diagsize** to 0 that will switch to default logging behavior when DB2 gets restarted, use the following command:

  ```
  db2 update dbm cfg using diagsize 0
  ```

Note: Starting with DB2 Version 9.7 Fix Pack 1, if the **diagsize** configuration parameter is set to a non-zero value and the **diagpath** configuration parameter is set to split the diagnostic data into separate directories, then the non-zero value of the **diagsize** configuration parameter specifies the total size of the combination of all rotating administration notification log files and all rotating diagnostic log files contained within a given split diagnostic data directory. For example, if a system with 4 database partitions has **diagsize** set to 1 GB and **diagpath** set to "$n" (split diagnostic data per database partition), the maximum total size of the combined notification and diagnostic logs can reach 4 GB (4 x 1 GB).

dir_cache - Directory cache support

This parameter determines whether the database, node and DCS directory files will be cached in memory.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
`Yes [Yes; No ]`

When allocated

- When an application issues its first connect, the application directory cache is allocated
- When a database manager instance is started (**db2start**), the server directory cache is allocated.

When freed

- When an the application process terminates, the application directory cache is freed
- When a database manager instance is stopped (**db2stop**), the server directory cache is freed.

The use of the directory cache reduces connect costs by eliminating directory file I/O and minimizing the directory searches required to retrieve directory information. There are two types of directory caches:

- An application directory cache that is allocated and used for each application process on the machine at which the application is running.
- A server directory cache that is allocated and used for some of the internal database manager processes.

For application directory caches, when an application issues its first connect, each directory file is read and the information is cached in private memory for this application. The cache is used by the application process on subsequent connect requests and is maintained for the life of the application process. If a database is not found in the application directory cache, the directory files are searched for the information, but the cache is not updated. If the application modifies a directory entry, the next connect within that application will cause the cache for this application to be refreshed. The application directory cache for other applications will not be refreshed. When the application process terminates, the cache is freed. (To refresh the directory cache used by a command line processor session, issue a **db2 terminate** command.)

For server directory caches, when a database manager instance is started (**db2start**), each directory file is read and the information is cached in the server memory. This cache is maintained until the instance is stopped (**db2stop**). If a directory entry is not found in this cache, the directory files are searched for the information. Normally, this server directory cache is never refreshed during the time the instance is running. However, an offline backup marks the server directory cache as invalid and will refresh the cache even with the instance running.

Recommendation: Use directory caching if your directory files do not change frequently and performance is critical.

In addition, on remote clients, directory caching can be beneficial if your applications issue several different connection requests. In this case, caching reduces the number of times a single application must read the directory files.

Directory caching can also improve the performance of taking database system monitor snapshots. In addition, you should explicitly reference the database name on the snapshot call, instead of using database aliases.

Note: Errors might occur when performing snapshot calls if directory caching is turned on and if databases are cataloged, uncataloged, created, or dropped after the database manager is started.

discover - Discovery mode

This parameter determines what kind of discovery requests, if any, the client can make.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
SEARCH [DISABLE, KNOWN, SEARCH]

From a client perspective, one of the following will occur:

- If **discover** = SEARCH, the client can issue search discovery requests to find DB2 server systems on the network. Search discovery provides a superset of the functionality provided by KNOWN discovery. If **discover** = SEARCH, both search and known discovery requests can be issued by the client.
- If **discover** = KNOWN, only known discovery requests can be issued from the client. By specifying some connection information for the administration server on a particular system, all the instance and database information about the DB2 system is returned to the client.
- If **discover** = DISABLE, discovery is disabled at the client.

The default discovery mode is SEARCH.

discover_inst - Discover server instance

This parameter specifies whether this instance can be detected by DB2 discovery.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Propagation class
Immediate

Default [range]
ENABLE [ENABLE, DISABLE]

The parameter's default, ENABLE, specifies that the instance can be detected, while DISABLE prevents the instance from being discovered.

fcm_num_buffers - Number of FCM buffers

This parameter specifies the number of 4 KB buffers that are used for internal communications (messages) both among and within database servers.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server or DB2 pureScale database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]

32-bit platforms
Automatic [895 - 65300]

64-bit platforms
Automatic [895 - 524288]

- Database server with local and remote clients: 1024
- Database server with local clients: 895
- Partitioned database server or DB2 pureScale database server with local and remote clients: 4096

Fast communication manager (FCM) buffers are used for both inter-member and intra-member communications by default.

Important: The default value of the **fcm_num_buffers** parameter is subject to change by the DB2 Configuration Advisor after initial database creation.

You can set both an initial value and the AUTOMATIC value for the **fcm_num_buffers** configuration parameter. When you set the parameter to AUTOMATIC, FCM monitors resource usage and can increase or decrease resources if they are not used within 30 minutes. The amount that resources are increased or decreased depends on the operating system. On Linux operating systems, the number of buffers can be increased only 25% above the starting value. If the database manager attempts to start an instance and cannot allocate the specified number of buffers, it decreases the number until it can start the instance.

If you want to set the **fcm_num_buffers** parameter to both a specific value and AUTOMATIC and you do not want the system controller thread to adjust resources

below the specified value, set the FCM_CFG_BASE_AS_FLOOR option of the **DB2_FCM_SETTINGS** registry variable to YES or TRUE. The **DB2_FCM_SETTINGS** registry variable value is adjusted dynamically.

If you are using multiple logical nodes, one pool of **fcm_num_buffers** buffers is shared by all the logical nodes on the same machine. You can determine the size of the pool by multiplying the value of the **fcm_num_buffers** parameter by the number of logical nodes on the physical machine. Examine the value that you are using; consider how many FCM buffers are allocated on a machine or machines with multiple logical nodes. If you have multiple logical nodes on the same machine, you might have to increase the value of the **fcm_num_buffers** parameter. The number of users on the system, the number of database partition servers on the system, or the complexity of the applications can cause a system to run out of message buffers.

fcm_num_channels - Number of FCM channels

This parameter specifies the number of FCM channels for each database partition.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server orDB2 pureScale database server with local and remote clients
- Satellite database server with local clients

Parameter type
Configurable online

Propagation class
Immediate

Default [range]

UNIX 32-bit platforms
Automatic, with a starting value of 256, 512 or 2048 [128 - 120000]

UNIX 64-bit platforms
Automatic, with a starting value of 256, 512 or 2048 [128 - 524288]

Windows 32-bit
Automatic, with a starting value 10000 [128 - 120000]

Windows 64-bit
Automatic, with a starting value of 256, 512 or 2048 [128 - 524288]

The default starting values for different types of servers are as follows:

- For database server with local and remote clients, the starting value is 512.
- For database server with local clients, the starting value is 256.
- For partitioned database environment servers with local and remote clients, the starting value is 2048.

Fast communication manager (FCM) buffers are used for both inter-member and intra-member communications by default. To enable

non-clustered database systems to use the FCM subsystem and the **fcm_num_channels** parameter, you had to set the **intra_parallel** parameter to YES

An FCM channel represents a logical communication end point between EDUs running in the DB2 engine. Both control flows (request and reply) and data flows (table queue data) rely on channels to transfer data between members.

When set to AUTOMATIC, FCM monitors channel usage, incrementally allocating and releasing resources as requirements change.

fcm_parallelism - Internode communication parallelism

This parameter specifies the degree of parallelism that is used for communication (both control messages and data flow) between members within a DB2 instance.

This parameter determines the number of sender and receiver fast communication manager conduit pairs. By default, an instance has only one pair: one sender and one receiver that handle all communication to and from other members in the instance.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
1 [1 - 8]

When this configuration parameter is set to 1, no parallelism is used. Updates to this parameter take effect only after you shut down and restart all members in an instance.

fed_noauth - Bypass federated authentication

This parameter determines whether federated authentication will be bypassed at the instance.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Propagation class
Immediate

Default [range]
No [Yes; No]

When **fed_noauth** is set to yes, **authentication** is set to server or server_encrypt, and **federated** is set to yes, then authentication at the instance is bypassed. It is assumed that authentication will happen at the data source. Exercise caution when **fed_noauth** is set to yes. Authentication is done at neither the client nor at DB2. Any user who knows the SYSADM authentication name can assume SYSADM authority for the federated server.

federated - Federated database system support

This parameter enables or disables support for applications submitting distributed requests for data managed by data sources (such as the DB2 Family and Oracle).

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
No [Yes; No]

federated_async - Maximum asynchronous TQs per query configuration parameter

This parameter determines the maximum number of asynchrony table queues (ATQs) in the access plan that the federated server supports. The mechanism of ATQs does not work in DB2 pureScale environments.

Configuration type
Database manager

Applies to

- Partitioned database server with local and remote clients when federation is enabled.

Parameter type
Configurable online

Default [range]
0 [0 to 32 767 inclusive, -1, ANY]

When ANY or -1 is specified, the optimizer determines the number of ATQs for the access plan. The optimizer assigns an ATQ to all eligible SHIP or remote pushdown operators in the plan. The value that is specified for DB2_MAX_ASYNC_REQUESTS_PER_QUERY server option limits the number of asynchronous requests.

Recommendation
The **federated_async** configuration parameter supplies the default or starting value for the special register and the bind option. You can override the value of this parameter by setting the value of the CURRENT FEDERATED ASYNCHRONY special register, **FEDERATED_ASYNCHRONY** bind option, or **FEDERATED_ASYNCHRONY** prepcompile option to a higher or a lower number.

If the special register or the bind option do not override the **federated_async** configuration parameter, the value of the parameter determines the maximum number of ATQs in the access plan that the federated server allows. If the special register or the bind option overrides this parameter, the value of the special register or the bind option determines the maximum number of ATQs in the plan.

Any changes to the **federated_async** configuration parameter affect dynamic statements as soon as the current unit of work commits. Subsequent dynamic statements recognize the new value automatically. A restart of the federated database is not needed. Embedded SQL packages are not invalidated nor implicitly rebound when the value of the **federated_async** configuration parameter changes.

If you want the new value of the **federated_async** configuration parameter to affect static SQL statements, you need to rebind the package.

fenced_pool - Maximum number of fenced processes

This parameter represents the number of threads cached in each **db2fmp** process for threaded **db2fmp** processes (processes serving threadsafe stored procedures and UDFs). For non-threaded **db2fmp** processes, this parameter represents the number of processes cached.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Default [range]
`-1` (**max_coordagents**), `Automatic [-1; 0-64 000]`

Unit of measure
Counter

Restrictions:

- If this parameter is updated dynamically, and the value is decreased, the database manager does not proactively terminate **db2fmp** threads or processes, instead it stops caching them as they are used in order to reduce the number of cached db2fmp's down to the new value.
- If this parameter is updated dynamically, and the value is increased, the database manager caches more **db2fmp** threads and processes when they are created.
- When this parameter is set to `-1`, the default, it assumes the value of the **max_coordagents** configuration parameter. Note that only the value of **max_coordagents** is assumed and not the automatic setting or behavior.
- When this parameter is set to `AUTOMATIC`, also the default:
  - The database manager allows the number of **db2fmp** threads and processes cached to increase based on the high water mark of coordinating agents. Specifically, the automatic behavior of this parameter allows it to grow depending on the maximum number of coordinating agents the database manager has ever registered, at the same time, since it started.

– The value assigned to this parameter represents a lower bound for the number of **db2fmp** threads and process to cache.

Recommendation: If your environment uses fenced stored procedures or user defined functions, then this parameter can be used to ensure that an appropriate number of **db2fmp** processes are available to process the maximum number of concurrent stored procedures and UDFs that run on the instance, ensuring that no new fenced mode processes need to be created as part of stored procedure and UDF execution.

If you find that the default value is not appropriate for your environment because an inappropriate amount of system resource is being given to **db2fmp** processes and is affecting performance of the database manager, the following procedure might be useful in providing a starting point for tuning this parameter:

```
fenced_pool = # of applications allowed to make stored procedure and
UDF calls at one time
```

If **keepfenced** is set to YES, then each **db2fmp** process that is created in the cache pool will continue to exist and will use system resources even after the fenced routine call has been processed and returned to the agent.

If **keepfenced** is set to NO, then non-threaded db2fmp processes will terminate when they complete execution, and there is no cache pool. Multithreaded **db2fmp** processes will continue to exist, but no threads will be pooled in these processes. This means that even when **keepfenced** is set to NO, you can have one threaded C **db2fmp** process and one threaded Java **db2fmp** process on your system.

In previous versions, this parameter was known as **maxdari**.

group_plugin - Group plug-in

This parameter specifies the name of the group plug-in library.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [any valid string]

By default, this value is null, and DB2 uses the operating system group lookup. The plug-in will be used for all group lookups. For non-root installations, if the DB2 userid and password plug-in library is used, the **db2rfe** command must be run before using your DB2 product.

health_mon - Health monitoring

This parameter allows you to specify whether you want to monitor an instance, its associated databases, and database objects according to various health indicators.

Important: This parameter is deprecated in Version 10.1 and might be removed in a future release. This parameter can still be used in releases before Version 10.1.

You cannot use this parameter in DB2 pureScale environments.

Configuration type
Database manager

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
`Off [On; Off ]`

Related Parameters

If **`health_mon`** is turned on, an agent will collect information about the health of the objects you have selected. If an object is considered to be in an unhealthy position, based on thresholds that you have set, notifications can be sent, and actions can be taken automatically. If **`health_mon`** is turned off, the health of objects will not be monitored.

You can use the CLP to select the instance and database objects that you want to monitor. You can also specify where notifications should be sent, and what actions should be taken, based on the data collected by the health monitor.

indexrec - Index re-creation time

This parameter indicates when the database manager attempts to rebuild invalid indexes, and whether or not any index build is redone during rollforward or high availability disaster recovery (HADR) log replay on the standby database.

Configuration type
Database and Database Manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]

UNIX Database Manager
`restart [restart; restart_no_redo; access; access_no_redo]`

Windows Database Manager
`restart [restart; restart_no_redo; access; access_no_redo]`

Database
Use system setting [`system; restart; restart_no_redo; access; access_no_redo`]

There are five possible settings for this parameter:

SYSTEM
Use system setting specified in the database manager configuration file to decide when invalid indexes are rebuilt, and whether any index build log records are to be redone during rollforward or HADR log replay.

Note: This setting is only valid for database configurations.

ACCESS
Invalid indexes are rebuilt when the underlying table is first accessed. Any fully logged index builds are redone during rollforward or HADR log replay. When HADR is started and an HADR takeover occurs, any invalid indexes are rebuilt after takeover when the underlying table is first accessed.

ACCESS_NO_REDO
Invalid indexes are rebuilt when the underlying table is first accessed. Any fully logged index build is not redone during rollforward or HADR log replay and those indexes are left invalid. When HADR is started and an HADR takeover takes place, any invalid indexes are rebuilt after takeover when the underlying table is first accessed. Access to the underlying tables on the new primary cause index rebuild, which causes log records to be written and then sent to the new standby, which in turn causes the indexes to be invalidated on the standby.

RESTART
The default value for **indexrec**. Invalid indexes are rebuilt when a **RESTART DATABASE** command is either explicitly or implicitly issued. Any fully logged index build will be redone during rollforward or HADR log replay. When HADR is started and an HADR takeover takes place, any invalid indexes will be rebuilt at the end of takeover.

Note: In a DB2 pureScale environment, indexes are rebuilt only during a group crash recovery, not as part of member crash recovery.

Note: When a database terminates abnormally while applications are connected to it, and the autorestart parameter is enabled, a **RESTART DATABASE** command is implicitly issued when an application connects to a database. If the command is not issued, the invalid indexes are rebuilt the next time the underlying table is accessed.

RESTART_NO_REDO
Invalid indexes are rebuilt when a **RESTART DATABASE** command is either explicitly or implicitly issued. Any fully logged index build is not redone during rollforward or HADR log replay and instead those indexes are rebuilt when rollforward completes or when HADR takeover takes place. Takeover causes index rebuild on underlying tables on the new primary, which causes log records to be written and then sent to the new standby, which in turn causes the indexes to be invalidated on the standby.

When a database terminates abnormally while applications are connected to it, and the autorestart parameter is enabled, a **RESTART DATABASE** command is implicitly issued when an application connects to a database. If the command is not issued, the invalid indexes are rebuilt the next time the underlying table is accessed.

Indexes can become invalid when fatal disk problems occur. If this happens to the data itself, the data could be lost. However, if this happens to an index, the index can be recovered by recreating it. If an index is rebuilt while users are connected to the database, two problems could occur:

- An unexpected degradation in response time might occur as the index file is re-created. Users accessing the table and using this particular index would wait while the index was being rebuilt.
- Unexpected locks might be held after index recreation, especially if the user transaction that caused the index to be re-created never performed a COMMIT or ROLLBACK.

Recommendation: The best choice for this option on a high-user server and if restart time is not a concern, would be to have the index rebuilt at **DATABASE RESTART** time as part of the process of bringing the database back online after a crash.

Setting this parameter to ACCESS or to ACCESS_NO_REDO result in a degradation of the performance of the database manager while the index is being re-created. Any user accessing that specific index or table would have to wait until the index is re-created.

If this parameter is set to RESTART, the time taken to restart the database is longer due to index re-creation, but normal processing would not be impacted once the database has been brought back online.

Note: At database recovery time, all SQL procedure executables on the file system that belong to the database being recovered are removed. If **indexrec** is set to RESTART, all SQL procedure executables are extracted from the database catalog and put back on the file system at the next connection to the database. If **indexrec** is not set to RESTART, an SQL executable is extracted to the file system only on first execution of that SQL procedure.

The difference between the RESTART and the RESTART_NO_REDO values, or between the ACCESS and the ACCESS_NO_REDO values, is only significant when full logging is activated for index build operations, such as **CREATE INDEX** and **REORG INDEX** operations, or for an index rebuild. You can activate logging by enabling the **logindexbuild** database configuration parameter or by enabling LOG INDEX BUILD when altering a table. By setting **indexrec** to either RESTART or ACCESS, operations involving a logged index build can be rolled forward without leaving the index object in an invalid state, which would require the index to be rebuilt at a later time.

instance_memory - Instance memory

This parameter specifies the maximum amount of memory that can be allocated for a database partition if you are using DB2 database products with memory usage restrictions or if you set it to a specific value. Otherwise, the AUTOMATIC setting allows instance memory to grow as needed.

Configuration type

Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Default [range]

32-bit platforms
Automatic [0 - 1 000 000]

64-bit platforms
Automatic [0 - 68 719 476 736]

DB2 Express® Edition and DB2 Express-C
Automatic [0 - 1 048 576]

DB2 Workgroup Server Edition
Automatic [0 - 4 194 304]

Unit of measure
Pages (4 KB)

When allocated
Not applicable

When freed
Not applicable

The default value of **instance_memory** is AUTOMATIC.The AUTOMATIC setting results in a value being computed at database partition activation. The computed value ranges between 75 percent and 95 percent of the physical RAM on the system - the larger the system, the higher the percentage. For DB2 database products with memory usage restrictions, the computed value is also limited by the maximum allowed by the product license. For database partition servers with multiple logical database partitions, this computed value is divided by the number of logical database partitions.

Starting with Version 9.7 Fix Pack 1 and Version 9.5 Fix Pack 5, the computed value for the AUTOMATIC setting does not enforce a limit on memory allocated across the instance for DB2 database products without memory usage restrictions. For Version 9.7 and Version 9.5 Fix Pack 4 or earlier, the computed value for the AUTOMATIC setting represents a limit for all DB2 database products.

Updating instance_memory dynamically

- Dynamic updates to **instance_memory** require an instance attachment. See the **ATTACH** command for details.
- For DB2 database products with memory usage restrictions, dynamic updates to **instance_memory** must indicate a value less than any license limit or AUTOMATIC. Otherwise, the update fails and the SQL5130N error message is returned.
- Dynamic updates to **instance_memory** must indicate a value less than the amount of physical RAM or AUTOMATIC . Otherwise, the update is deferred until the next **db2start** is issued and the SQL1362W warning message is returned.
- Dynamic updates to **instance_memory** must indicate a value larger than the current amount of in-use instance memory. Otherwise, the update is deferred until the instance is restarted, and the SQL1362W warning message is returned. The amount of in-use instance memory can be determined by subtracting the *Cached memory* value from *Current usage*

value in the output of the **db2pd -dbptnmem** command. The minimum value would be the highest in-use instance memory across all database partitions.

- If **instance_memory** is set to a value greater than the amount of physical RAM, the next **db2start** command that you issue will fail and return the SQL1220N error message.
- If **instance_memory** is dynamically updated to AUTOMATIC, the value is recalculated immediately.

Restriction for instances in partitioned database environments

You should not use of a specific value for **instance_memory** in partitioned database environments. Using a specific value for **instance_memory** is not recommended in partitioned database environments because the **instance_memory** is a database manager configuration parameter and it is not possible to specify different values for different database partitions. This makes it difficult to establish a setting suitable for all database partitions because they might have different memory requirements.

Controlling DB2 Memory consumption:

DB2 memory consumption varies depending on workload and configuration. In addition to this, self-tuning of **database_memory** becomes a factor if it is enabled. Self-tuning of **database_memory** is enabled when **database_memory** is set to AUTOMATIC and the self-tuning memory manager (STMM) is active.

If the instance is running on a DB2 database product without memory usage restrictions and **instance_memory** is set to AUTOMATIC, an **instance_memory** limit is not enforced. The database manager allocates system memory as needed. If self-tuning of **database_memory** is enabled, STMM updates the configuration to achieve optimal performance while monitoring available system memory. This monitoring of available memory ensures that system memory is not over-committed

If the instance is running on a DB2 database product with memory usage restrictions or **instance_memory** is set to a specific value, an **instance_memory** limit is enforced. The database manager allocates system memory up to this limit, the application can receive memory allocation errors when this limit is reached. Additional consideration are as follows:

- If self-tuning of **database_memory** is enabled and **instance_memory** is set to a specific value, STMM updates the configuration to achieve optimal performance while maintaining sufficient free instance memory. This ensures that enough instance memory is available to satisfy volatile memory requirements. System memory is not monitored.
- If self-tuning of **database_memory** is enabled and **instance_memory** is set to AUTOMATIC, this is the case where an **instance_memory** limit is enforced for DB2 database product with memory usage restrictions, STMM updates the configuration to achieve optimal performance while monitoring available system memory and maintaining sufficient free instance memory.

Monitoring Instance Memory usage

Use the **db2pd -dbptnmem** command to show details on instance memory usage.

Use the new ADMIN_GET_MEM_USAGE table function to get the total instance memory consumption by a DB2 instance for a specific database partition, or for all database partitions. This table function also returns the current upper bound value.

When fast communication manager (FCM) shared memory is allocated, each local database partition's share of the overall FCM shared memory size for the system is accounted for in that database partition's **instance_memory** usage.

intra_parallel - Enable intrapartition parallelism

This parameter specifies whether the database manager can use intrapartition query parallelism.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
NO (0) [SYSTEM (-1), NO (0), YES (1)]

A value of -1 causes the parameter value to be set to YES or NO based on the hardware on which the database manager is running.

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Note:

- Parallel index creation does not use this configuration parameter.
- If you change this parameter value, packages might be rebound to the database, and some performance degradation might occur.
- The **intra_parallel** setting can be overridden in an application by a call to the ADMIN_SET_INTRA_PARALLEL procedure. Both the **intra_parallel** setting and the value set in an application by the ADMIN_SET_INTRA_PARALLEL procedure can be overridden in a workload by setting the MAXIMUM DEGREE attribute in a workload definition.

java_heap_sz - Maximum Java interpreter heap size

This parameter determines the maximum size of the heap that is used by the Java interpreter started to service Java DB2 stored procedures and UDFs.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]

HP-UX
4096 [0 - 524 288]

All other operating systems
2048 [0 - 524 288]

Unit of measure
Pages (4 KB)

When allocated
When a Java stored procedure or UDF starts

When freed
When the db2fmp process (fenced) or the db2agent process (trusted) terminates.

There is one heap per db2fmp process running a Java stored procedure. For multithreaded db2fmp processes, multiple applications using threadsafe fenced routines are serviced from a single heap. In all situations, only the processes that run Java UDFs or stored procedures ever allocate this memory. On partitioned database systems, the same value is used at each database partition.

XML data is materialized when passed to stored procedures as IN, OUT, or INOUT parameters. When you are using Java stored procedures, the heap size might need to be increased based on the quantity and size of XML arguments, and the number of external stored procedures that are being executed concurrently.

jdk_path - Software Developer's Kit for Java installation path

This parameter specifies the directory under which the Software Developer's Kit (SDK) for Java, to be used for running Java stored procedures and user-defined functions, is installed. The **CLASSPATH** and other environment variables used by the Java interpreter are computed from the value of this parameter.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [Valid path]

If the SDK for Java was installed with your DB2 product, this parameter is set properly. However, if you reset the database manager (dbm cfg) parameter, you need to specify where the SDK for Java is installed.

keepfenced - Keep fenced process

This parameter indicates whether or not a fenced mode process is kept after a fenced mode routine call is complete. Fenced mode processes are created as

separate system entities in order to isolate user-written fenced mode code from the database manager agent process. This parameter is only applicable on database servers.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

s

Parameter type
Configurable

Default [range]
`Yes [Yes; No ]`

If **keepfenced** is set to `No`, and the routine being executed is not threadsafe, a new fenced mode process is created and destroyed for each fenced mode invocation. If **keepfenced** is set to `no`, and the routine being executed is threadsafe, the fenced mode process persists, but the thread created for the call is terminated. If **keepfenced** is set to `yes`, a fenced mode process or thread is reused for subsequent fenced mode calls. When the database manager is stopped, all outstanding fenced mode processes and threads will be terminated.

Setting this parameter to `yes` will result in additional system resources being consumed by the database manager for each fenced mode process that is activated, up to the value contained in the **fenced_pool** parameter. A new process is only created when no existing fenced mode process is available to process a subsequent fenced routine invocation. This parameter is ignored if **fenced_pool** is set to `0`.

Recommendation: In an environment in which the number of fenced mode requests is large relative to the number of non-fenced mode requests, and system resources are not constrained, then this parameter can be set to `yes`. This will improve the fenced mode process performance by avoiding the initial fenced mode process creation processing time since an existing fenced mode process will be used to process the call. In particular, for Java routines, this will save the cost of starting the Java Virtual Machine (JVM), a very significant performance improvement.

For example, in an OLTP debit-credit banking transaction application, the code to perform each transaction could be performed in a stored procedure which executes in a fenced mode process. In this application, the main workload is performed out of fenced mode processes. If this parameter is set to `no`, each transaction incurs the additional processing time of creating a new fenced mode process, resulting in a significant performance reduction. If, however, this parameter is set to `yes`, each transaction would try to use an existing fenced mode process, which would avoid this additional processing time.

In previous versions of DB2, this parameter was known as **keepdari**.

local_gssplugin - GSS API plug-in used for local instance level authorization

This parameter specifies the name of the default GSS API plug-in library to be used for instance level local authorization when the value of the **authentication** database manager configuration parameter is set to GSSPLUGIN or GSS_SERVER_ENCRYPT.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [any valid string]

max_connections - Maximum number of client connections

This parameter indicates the maximum number of client connections allowed per member.

Configuration type
Database manager

Parameter type
Configurable online

Applies to

- Database server with local and remote clients
- Database server with local clients
- Database Server or Connect server with local and remote clients (for **max_connections**, **max_coordagents**, **num_initagents**, **num_poolagents**, and also **federated_async**, if you are using a Federated environment)

Default [range]
-1 and AUTOMATIC (**max_coordagents**) [-1 and AUTOMATIC, 1 - 64000]

A setting of -1 means that the value associated with **max_coordagents** will be used, not the automatic setting or behavior. AUTOMATIC means that the database manager picks the value for this parameter using whatever technique works best. AUTOMATIC is an ON/OFF switch in the configuration file and is independent of the value, hence both -1 and AUTOMATIC can be the default setting.

For details, see: "Restrictions and behavior when configuring max_coordagents and max_connections" on page 659.

The Concentrator

The Concentrator is OFF when **max_connections** is equal to or less than **max_coordagents**. The Concentrator is ON when **max_connections** is greater than **max_coordagents**.

This parameter controls the maximum number of client applications that can be connected to a member in the instance. Typically, each application is assigned a coordinator agent. The agent facilitates the operations between the application and the database. When the default value for this parameter is used, the Concentrator feature is not activated. As a result, each agent operates within its own private memory and shares database manager and database global resources, such as the buffer pool, with other agents. When the parameter is set to a value greater than the default, the Concentrator feature is activated.

Usage

The maximum number of client connections can be larger than the value set by the **max_connections** configuration parameter if the user holds one of the following authorities:

- SYSADM
- SYSCTRL
- SYSMAIN

Note: Bypassing the maximum number of client connections set by the **max_connections** configuration parameter is useful for many reasons including the following tasks:

- Administrative tasks
- Capturing snapshot information
- Critical troubleshooting tasks
- Maintenance tasks (such as forcing users off the system)

max_connretries - Node connection retries

This parameter specifies the maximum number of times an attempt will be made to establish a network connection between two DB2 members.

Configuration type
Database manager

Applies to
Partitioned database server with local and remote clients

DB2 pureScale server

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
5 [0-100]

If the attempt to establish communication between two DB2 members fails (for example, the value specified by the **conn_elapse** parameter is reached), **max_connretries** specifies the number of connection retries that can be made to a DB2 member. If the value specified for this parameter is exceeded, an error is returned.

max_coordagents - Maximum number of coordinating agents

This parameter is used to limit the number of coordinating agents.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Default [range]
`200, Automatic [-1; 0-64 000]`

A setting of `-1` translates into a value of `200`.

For details, see: "Restrictions and behavior when configuring max_coordagents and max_connections" on page 659.

The Concentrator

When the Concentrator is OFF, that is, when **max_connections** is equal to or less than **max_coordagents**, this parameter determines the maximum number of coordinating agents that can exist at one time on a server node.

One coordinating agent is acquired for each local or remote application that connects to a database or attaches to an instance. Requests that require an instance attachment include **CREATE DATABASE**, **DROP DATABASE**, and Database System Monitor commands.

When the Concentrator is ON, that is, when **max_connections** is greater than **max_coordagents**, there might be more connections than coordinator agents to service them. An application is in an active state only if there is a coordinator agent servicing it. Otherwise, the application is in an inactive state. Requests from an active application will be serviced by the database coordinator agent (and subagents in SMP or MPP configurations). Requests from an inactive application will be queued until a database coordinator agent is assigned to service the application, when the application becomes active. As a result, this parameter might be used to control the load on the system.

Usage

The maximum number of coordinating agents can be larger than the value set by the **max_coordagents** configuration parameter if the user holds one of the following authorities:

- SYSADM
- SYSCTRL
- SYSMAIN

Note: Bypassing the maximum number of coordinating agents set by the **max_coordagents** configuration parameter is useful for many reasons including the following tasks:

- Administrative tasks
- Capturing snapshot information
- Critical troubleshooting tasks
- Maintenance tasks (such as forcing users off the system)

max_querydegree - Maximum query degree of parallelism

This parameter specifies the maximum degree of intrapartition parallelism that is used for any SQL statement executing on this instance of the database manager. An SQL statement will not use more than this number of parallel operations within a database partition when the statement is executed.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Statement boundary

Default [range]
`-1 (ANY) [ANY, 1 - 32 767]` (`ANY` means system-determined)

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

The **intra_parallel** configuration parameter must be set to `YES` to enable the database partition to use intrapartition parallelism for SQL statements. The **intra_parallel** parameter is no longer required for parallel index creation.

The default value for this configuration parameter is `-1`. This value means that the system uses the degree of parallelism determined by the optimizer; otherwise, the user-specified value is used.

Note: The degree of parallelism for an SQL statement can be specified at statement compilation time using the CURRENT DEGREE special register or the **DEGREE** bind option.

The maximum query degree of parallelism for an active application can be modified using the **SET RUNTIME DEGREE** command. The actual runtime degree used is the lower of:

- **max_querydegree** configuration parameter
- Application runtime degree
- SQL statement compilation degree

This configuration parameter applies to queries only.

max_time_diff - Maximum time difference between members

This parameter specifies the maximum time difference that is permitted between members in a DB2 pureScale environment that are listed in the node configuration file.

Configuration type
Database manager

Applies to
Members with local and remote clients

Parameter type
Configurable

Default [range]

In DB2 pureScale environments
1 [1 - 1 440]

Outside of DB2 pureScale environments
60 [1 - 1 440]

Unit of measure
Minutes

Each member has its own system clock. The time difference between two or more member system clocks is checked periodically. If the time difference between the system clocks is more than the amount specified by the **max_time_diff** parameter, warnings are logged in the db2diag log files.

In a DB2 pureScale environment, to ensure that members do not drift out of sync with each other, a Network Time Protocol (NTP) setup is required and periodically verified on each member. If the NTP daemon is not detected, warnings are logged in the db2diag log files.

The SQL1473N error message is returned in partitioned database environments where the system clock is compared to the virtual time stamp (VTS) saved in the SQLOGCTL.LFH log control file. If the time stamp in the .LFH log control file is less than the system time, the time in the database log is set to the VTS until the system clock matches the VTS.

DB2 database manager uses Coordinated Universal Time (UTC), so different time zones are not a consideration when you set the **max_time_diff** parameter. UTC is the same as Greenwich Mean Time.

maxagents - Maximum number of agents

This parameter has been deprecated since Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the database manager in DB2 Version 9.5 or later releases.

Note: The following information applies only to pre-Version 9.5 data servers and clients.

This parameter indicates the maximum number of database manager agents, whether coordinator agents or subagents, available at any given time to accept application requests

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]

200 [1 - 64 000]

400 [1 - 64 000] on partitioned database server with local and remote clients

Unit of measure

Counter

If you want to limit the number of coordinating agents, use the **max_coordagents** parameter.

This parameter can be useful in memory constrained environments to limit the total memory usage of the database manager, because each additional agent requires additional memory.

Recommendation: The value of **maxagents** should be at least the sum of the values for **maxappls** in each database allowed to be accessed concurrently. If the number of databases is greater than the **numdb** parameter, then the safest course is to use the product of **numdb** with the largest value for **maxappls**.

Each additional agent requires some additional processing resources that are allocated at the time the database manager is started.

If you are encountering memory errors when trying to connect to a database, try making the following configuration adjustments:

- In a non-partitioned database environment with no intra-query parallelism enabled, increase the value of the **maxagents** database configuration parameter.
- In a partitioned database environment or an environment where intra-query parallelism is enabled, increase the larger of **maxagents** or **max_coordagents**.

maxcagents - Maximum number of concurrent agents

This parameter has been deprecated since Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the database manager in DB2 Version 9.5 or later releases.

Note: The following information applies only to pre-Version 9.5 data servers and clients.

This parameter is used to control the load on the system during periods of high simultaneous application activity by limiting the maximum number of database manager agents that can be concurrently executing a database manager transaction

Configuration type

Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type

Configurable

Default [range]

-1 (**max_coordagents**) [-1; 1 - **max_coordagents**]

Unit of measure
Counter

This parameter does not limit the number of applications that can have connections to a database. It only limits the number of database manager agents that can be processed concurrently by the database manager at any one time, thereby limiting the usage of system resources during times of peak processing. For example, you might have a system requiring a large number of connections but with a limited amount of memory to serve those connections. Adjusting this parameter can be useful in such an environment, where a period of high simultaneous activity could cause excessive operating system paging.

A value of `-1` indicates that the limit is **max_coordagents**.

Recommendation: In most cases the default value for this parameter will be acceptable. In cases where the high concurrency of applications is causing problems, you can use benchmark testing to tune this parameter to optimize the performance of the database.

mon_heap_sz - Database system monitor heap size

This parameter determines the amount of the memory, in pages, to allocate for database system monitor data. Memory is allocated from the monitor heap when database monitoring activities are performed such as turning on monitor switches, resetting monitor data, activating an event monitor or sending monitor events to an active event monitor.

With Version 9.5, this database configuration parameter has a default value of `AUTOMATIC`, meaning that the monitor heap can increase as needed until the **instance_memory** limit is reached.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Default [range]
`Automatic [0 - 60 000]`

Unit of measure
Pages (4 KB)

When allocated
When the database manager is started with the **db2start** command

When freed
When the database manager is stopped with the **db2stop** command

A value of zero prevents the database manager from collecting database system monitor data.

Recommendation: The amount of memory required for monitoring activity depends on the number of monitoring applications (applications taking snapshots or event monitors), which switches are set, and the level of database activity.

If the configured memory in this heap runs out and no more unreserved memory is available in the instance shared memory region, one of the following events will occur:

- When the first application connects to the database for which this event monitor is defined, an error message is written to the administration notification log.
- If an event monitor being started dynamically using the SET EVENT MONITOR statement fails, an error code is returned to your application.
- If a monitor command or API subroutine fails, an error code is returned to your application.
- If an application needs to send an event monitor record but cannot allocate the record out of monitor heap, the application may be blocked until a record is allocated.

nodetype - Instance node type

This parameter specifies the type of instance (the type of database manager) created by DB2 products installed on your machine.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Informational

The possible values that are returned by this parameter and the products associated with that node type are shown in the following list:

- **Database server with local and remote clients** - a DB2 server product, supporting local and remote Data Server Runtime Clients, and capable of accessing other remote database servers.
- **Client** - a Data Server Runtime Client capable of accessing remote database servers.
- **Database server with local clients** - a DB2 relational database management system, supporting local Data Server Runtime Clients and capable of accessing other, remote database servers.
- **Partitioned database server with local and remote clients** - a DB2 server product, supporting local and remote Data Server Runtime Clients, and capable of accessing other remote database servers, and capable of parallelism.

notifylevel - Notify level

This parameter specifies the type of administration notification messages that are written to the administration notification log.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
3 [0 - 4]

On Linux and UNIX operating systems, the administration notification log is a text file called *instance*.nfy. On Windows, all administration notification messages are written to the Event Log. The errors can be written by DB2, the Health Monitor, the Capture and Apply programs, and user applications.

Valid values for this parameter are:

- 0 - No administration notification messages captured. (This setting is not recommended.)
- 1 - Fatal or unrecoverable errors. Only fatal and unrecoverable errors are logged. To recover from some of these conditions, you might need assistance from DB2 service.
- 2 - Immediate action required. Conditions are logged that require immediate attention from the system administrator or the database administrator. If the condition is not resolved, it could lead to a fatal error. Notification of very significant, non-error activities (for example, recovery) might also be logged at this level. This level will capture Health Monitor alarms. Informational messages will be shown at this level.
- 3 - Important information, no immediate action required. Conditions are logged that are non-threatening and do not require immediate action but might indicate a non-optimal system. This level will capture Health Monitor alarms, Health Monitor warnings, and Health Monitor attentions.

Usage Notes

- The administration notification log includes messages having values up to and including the value of **notifylevel**. For example, setting **notifylevel** to 3 will cause the administration notification log to include messages applicable to levels 1, 2, and 3.
- For a user application to be able to write to the notification file or Windows Event Log, it must call the db2AdminMsgWrite API.
- You might want to increase the value of this parameter to gather additional problem determination data to help resolve a problem. Note that you must set **notifylevel** to a value of 2 or higher for the Health Monitor to send any notifications to the contacts defined in its configuration.
- In specific circumstances, to display high importance messages, DB2 will override the **notifylevel** setting

num_initagents - Initial number of agents in pool

This parameter determines the initial number of idle agents that are created in the agent pool at **db2start** time.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Default [range]
0 [0-64 000]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

The database manager always starts the **num_initagents** idle agents as part of the **db2start** command, except if the value of this parameter is greater than **num_poolagents** during start up and **num_poolagents** is not set to AUTOMATIC. In this case, the database manager only starts the **num_poolagents** idle agents since there is no reason to start more idle agents than can be pooled.

num_initfenced - Initial number of fenced processes

This parameter indicates the initial number of non-threaded, idle **db2fmp** processes that are created in the **db2fmp** pool at **START DBM** time.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Default [range]
0 [0-64 000]

Setting this parameter will reduce the initial startup time for running non-threadsafe C and Cobol routines. This parameter is ignored if **keepfenced** is not specified.

It is much more important to set **fenced_pool** to an appropriate size for your system than to start up a number of **db2fmp** processes at **START DBM** time.

In previous versions, this parameter was known as **num_initdaris**.

num_poolagents - Agent pool size

This parameter sets the maximum size of the idle agent pool.

Configuration type
Database manager

Applies to

- Database server with local and remote clients

- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Default
`100, AUTOMATIC [-1, 0 - 64000]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

This configuration parameter is set to `AUTOMATIC` with a value of `100` as the default. A setting of `-1` is still supported, and translates into a value of `100`. When this parameter is set to `AUTOMATIC`, the database manager automatically manages the number of idle agents to pool. Typically, this means that once an agent completes its work, it is not terminated, but becomes idle for a period of time. Depending on the workload and type of agent, it might be terminated after a certain amount of time.

When using `AUTOMATIC`, you can still specify a value for the **num_poolagents** configuration parameter. Additional idle agents will always be pooled when the current number of pooled idle agents is less than or equal to the value that you specified.

Examples

num_poolagents is set to 100 and AUTOMATIC
As an agent becomes free, it is added to the idle agent pool, where at some point the database manager evaluates whether it should be terminated or not. At the time when the database manager considers terminating the agent, if the total number of idle agents pooled is greater than 100, this agent will be terminated. If there are less than 100 idle agents, the idle agent will remain awaiting work. Using the `AUTOMATIC` setting allows additional idle agents beyond 100 to be pooled, which might be useful during periods of heavier system activity when the frequency of work can fluctuate on a larger scale. For cases where there are likely to be less than 100 idle agents at any given time, agents are guaranteed to be pooled. Periods of light system activity can benefit from this by incurring a less start up cost for new work.

num_poolagents is configured dynamically
If the parameter value is increased to a value greater than the number of pooled agents, the effects are immediate. As new agents become idle, they are pooled. If the parameter value is decreased, the database manager does not immediately reduce the number of agents in the pool. Rather, the pool size remains as it is, and agents are terminated as they are used and become idle again-gradually reducing the number of agents in the pool to the new limit.

Recommendation

For most environments the default of `100` and `AUTOMATIC` will be sufficient. If you have a specific workload where you feel too many agents are being created and terminated, you can consider increasing the value of **num_poolagents** while leaving the parameter set to `AUTOMATIC`.

numdb - Maximum number of concurrently active databases including host and System i databases

This parameter specifies the number of local databases that can be concurrently active, or the maximum number of different database aliases that can be cataloged on a DB2 Connect server.

This parameter is overloaded.

The effect of the **numdb** database manager configuration parameter depends on the environment in which the parameter is used.

- For a DB2 Connect environment, you can specify the maximum number of database aliases that can be cataloged on a DB2 Connect server using the **numdb** database manager configuration parameter.
- Outside of a DB2 Connect environment, you can specify the number of local databases that can be concurrently active using the **numdb** database manager configuration parameter.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with only local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]

UNIX DB2 pureScale environment
32 [1 - 200]

UNIX Database server with local and remote clients
32 [1 - 256]

UNIX Database server with only local clients
8 [1 - 256]

Windows Database server with local and remote clients
32 [1 - 256]

Windows Database server with only local clients
3 [1 - 256]

Unit of measure
Counter

Usage notes

- It is important to remember when setting or updating this parameter that each database takes up storage, and each active database requires an additional new shared memory segment.
- Set **numdb** to the default value if the number of databases being run on the instance is less than the default.
- Use of the **numdb** configuration parameter must be coordinated with the **db2_database_cf_memory** and **cf_db_mem_sz** configuration parameters.
- Changing the **numdb** parameter can impact the total amount of memory allocated. As a result, frequent updates to this parameter are not recommended. Plan the

configurations for all parameters that are related to memory allocation for a database or an application connected to that database, and change them all at the same time.

- In addition to the **numdb** limit in a DB2 pureScale environment, there is a maximum number of database activations across all members in an instance. This maximum number is 512 database activations. As an example, if each member in a four member DB2 pureScale environment activated 200 databases, that is a total of 800 database member activations. Since 800 database activations exceeds the maximum upper limit, an error is returned.
- In a multiple database environment, if member crash recovery (MCR) is required, the number of databases that will be recovered in parallel on each member is set by the value of the **numdb** configuration parameter or the **DB2_MCR_RECOVERY_PARALLELISM_CAP** registry variable, whichever value is smaller.

query_heap_sz - Query heap size

This parameter specifies the maximum amount of memory that can be allocated for the query heap, ensuring that an application does not consume unnecessarily large amounts of virtual memory within an agent.

Important: This parameter is deprecated in Version 9.5 and might be removed in a future release. This parameter can still be used in pre-Version 9.5 data servers and clients. In Version 9.5 and later releases, the value specified for this configuration parameter is ignored.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
`1 000 [2 - 524 288 ]`

Unit of measure
Pages (4 KB)

When allocated
When an application (either local or remote) connects to the database

When freed
When the application disconnects from the database, or detaches from the instance

A query heap is used to store each query in the agent's private memory. The information for each query consists of the input and output SQLDA, the statement text, the SQLCA, the package name, creator, section number, and consistency token.

The query heap is also used for the memory allocated for blocking cursors. This memory consists of a cursor control block and a fully resolved output SQLDA.

The initial query heap allocated will be the same size as the application support layer heap, as specified by the **aslheapsz** parameter. The query heap size must be

greater than or equal to two (2), and must be greater than or equal to the **aslheapsz** parameter. If this query heap is not large enough to handle a given request, it will be reallocated to the size required by the request (not exceeding **query_heap_sz**). If this new query heap is more than 1.5 times larger than **aslheapsz**, the query heap will be reallocated to the size of **aslheapsz** when the query ends.

Recommendation: In most cases the default value will be sufficient. As a minimum, you should set **query_heap_sz** to a value at least five times larger than **aslheapsz**. This will allow for queries larger than **aslheapsz** and provide additional memory for three or four blocking cursors to be open at a given time.

If you have very large LOBs, you might need to increase the value of this parameter so the query heap will be large enough to accommodate those LOBs.

release - Configuration file release level

This parameter specifies the release level of the configuration file.

Configuration type
Database manager, Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Informational

rstrt_light_mem - Restart light memory configuration parameter

This parameter specifies the maximum amount of memory that is allocated and reserved on a host for restart light recovery purposes. The amount is a percentage of **instance_memory** configuration parameter.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
`AUTOMATIC` [1 - 10 for `AUTOMATIC`; the user-defined range is `1 - 50`]

Units of measure
Percentage of instance memory

When allocated
When the instance is started

When freed
When the instance is stopped

Having **rstrt_light_mem** set to AUTOMATIC means that when the instance is started, the DB2 database manager automatically calculates a fixed upper bound for the amount of memory to be pre-allocated and reserved for restart light recovery purposes. The specified amount of memory is reserved to accommodate failed members that need to be restarted on a host other than their home host (restart light mode). The DB2 database manager calculates the appropriate value based on the current setting for **instance_memory** and **numdb** (maximum number of currently active databases) as well as the number of members on the host . The automatically calculated value ranges between 1 and 10 percent of the instance memory limit and is included in the total amount of instance memory. Users can also explicitly set **rstrt_light_mem** to a value between 1 and 50 percent of the instance memory limit. Additional memory could result in an improvement in the recovery time, especially when there are multiple databases that need to be recovered. The additional memory will also result in less memory being available for normal work, reducing the throughput the members can handle.

The parameter is configurable but not online. Once the reserved recovery memory has been allocated when the DB2 instance is started, the amount of memory is fixed and will not change unless the configuration value is changed and the DB2 instance is globally stopped and started again.

To display information about the total amount of memory allocated on a host, use **db2pd** with the new **-totalmem** option. This information will also include the amount of restart light memory allocated on a host. Only information about the current host being accessed is returned, but **db2pd** can be run on separate hosts in parallel.

resync_interval - Transaction resync interval

This parameter specifies the time interval in seconds for which a transaction manager (TM), resource manager (RM) or sync point manager (SPM) should retry the recovery of any outstanding indoubt transactions found in the TM, the RM, or the SPM.

This parameter is applicable when you have transactions running in a distributed unit of work (DUOW) environment. This parameter also applies to recovery of federated database systems.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
180 [1 - 60 000]

Unit of measure
Seconds

Recommendation: If, in your environment, indoubt transactions will not interfere with other transactions against your database, you might want to increase the value of this parameter. If you are using a DB2 Connect gateway to access DRDA2

application servers, you should consider the effect indoubt transactions might have at the application servers even though there will be no interference with local data access. If there are no indoubt transactions, the performance impact will be minimal.

rqrioblk - Client I/O block size

This parameter specifies the block size at the Data Server Runtime Client when a blocking cursor is opened.

Configuration type
Database manager

Applies to

- Database server with remote clients
- Client
- Partitioned database server with remote clients

Parameter type
Configurable

Default [range]
32 767 [4 096 - 65 535]

Unit of measure
Bytes

The memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the Data Server Runtime Client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks might yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4 096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks might also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application.

sheapthres - Sort heap threshold

This parameter is an instance-wide soft limit on the total amount of memory that can be consumed by private sorts at any given time. When the total private sort memory consumption for an instance reaches this limit, the memory allocated for additional incoming private sort requests is considerably reduced.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- OLAP functions

Parameter type
Configurable online

Propagation class
Immediate

Default [Range]

UNIX 32-bit platforms
`0 [0, 250 - 2097152]`

Windows 32-bit platforms
`0 [0, 250 - 2097152]`

64-bit platforms
`0 [0, 250 - 2147483647]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

Examples of operations that use the sort heap include: sorts, hash joins, dynamic bitmaps (used for index ANDing and Star Joins), and table in-memory operations.

Explicit definition of the threshold prevents the database manager from using excessive amounts of memory for large numbers of sorts.

There is no reason to increase the value of this parameter when moving from a non-partitioned to a partitioned database environment. Once you have tuned the database and database manager configuration parameters on a single database partition environment, the same values will in most cases work well in a partitioned database environment. The only way to set this parameter to different values on different nodes or database partitions is to create more than one DB2 instance. This will require managing different DB2 databases over different database partition groups. Such an arrangement defeats the purpose of many of the advantages of a partitioned database environment.

When the instance-level **sheapthres** is set to `0`, then the tracking of sort memory consumption is done at the database level only and memory allocation for sorts is constrained by the value of the database-level **sheapthres_shr** configuration parameter.

Automatic tuning of **sheapthres_shr** is allowed only when the database manager configuration parameter **sheapthres** is set to `0`.

This parameter will not be dynamically updatable if any of the following are true:

- The starting value for **sheapthres** is `0` and the target value is a value different from 0.
- The starting value for **sheapthres** is a value different from 0 and the target value is 0.

Recommendation: Ideally, you should set this parameter to a reasonable multiple of the largest **sortheap** parameter you have in your database manager instance. This parameter should be **at least** two times the largest **sortheap** defined for any database within the instance.

If you are doing private sorts and your system is not memory constrained, an ideal value for this parameter can be calculated using the following steps:

1. Calculate the typical sort heap usage for each database:

```
  (typical number of concurrent agents running against the database)
* (sortheap, as defined for that database)
```

2. Calculate the sum of the preceding results, which provides the total sort heap that could be used under typical circumstances for all databases within the instance.

You should use benchmarking techniques to tune this parameter to find the proper balance between sort performance and memory usage.

You can use the database system monitor to track the sort activity, using the post threshold sorts (**post_threshold_sorts**) monitor element.

spm_log_file_sz - Sync point manager log file size

This parameter identifies the sync point manager (SPM) log file size in 4 KB pages.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
256 [4 - 1000]

Unit of measure
Pages (4 KB)

The log file is contained in the spmlog subdirectory under sqllib and is created the first time SPM is started.

Recommendation: The sync point manager log file size should be large enough to maintain performance, but small enough to prevent wasted space. The size required depends on the number of transactions using protected conversations, and how often COMMIT or ROLLBACK is issued.

To change the size of the SPM log file:

1. Determine that there are no indoubt transactions by using the **LIST DRDA INDOUBT TRANSACTIONS** command.
2. If there are none, stop the database manager.
3. Update the database manager configuration with a new SPM log file size.
4. Go to the $HOME/sqllib directory and issue rm -fr spmlog to delete the current SPM log. (Note: This shows the AIX command. Other systems might require a different remove or delete command.)
5. Start the database manager. A new SPM log of the specified size is created during the startup of the database manager.

spm_log_path - Sync point manager log file path

This parameter specifies the directory where the sync point manager (SPM) logs are written.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
NULL [any valid path or device]

If this parameter is null, by default, the logs are written to the `sqllib/spmlog` directory, which, in a high-volume transaction environment, can cause an I/O bottleneck. Use this parameter to have the SPM log files placed on a faster disk than the current `sqllib/spmlog` directory. This allows for better concurrency among the SPM agents.

Note: If SPM is enabled then the default directory will be created if it does not yet exist. To enable SPM, the configuration parameter **spm_name** must be set.

spm_max_resync - Sync point manager resync agent limit

This parameter identifies the number of agents that can simultaneously perform resync operations.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
20 [10 - 256]

spm_name - Sync point manager name

This parameter identifies the name of the sync point manager (SPM) instance to the database manager.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default
Derived from the TCP/IP hostname

srvcon_auth - Authentication type for incoming connections at the server

This parameter specifies how and where user authentication is to take place when handling incoming connections at the server; it is used to override the current authentication type.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
NOT_SPECIFIED [NOT_SPECIFIED; CLIENT; SERVER; SERVER_ENCRYPT; DATA_ENCRYPT; DATA_ENCRYPT_CMP; KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

If a value is not specified, DB2 uses the value of the **authentication** database manager configuration parameter.

For a description of each authentication type, see "authentication - Authentication type" on page 669.

srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server

This parameter specifies the GSS API plug-in libraries that are supported by the database server. It handles incoming connections at the server when the **srvcon_auth** parameter is specified as KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN or GSS_SERVER_ENCRYPT, or when **srvcon_auth** is not specified, and **authentication** is specified as KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN or GSS_SERVER_ENCRYPT.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [any valid string]

By default, the value is null. If the authentication type is GSSPLUGIN and this parameter is NULL, an error is returned. If the authentication type is KERBEROS and this parameter is NULL, the DB2 supplied kerberos module or library is used. This parameter is not used if another authentication type is used.

When the authentication type is KERBEROS and the value of this parameter is not NULL, the list must contain exactly one Kerberos plug-in, and that plug-in is used for authentication (all other GSS plug-ins in the list are ignored). If there is more than one Kerberos plug-in, an error is returned.

Each GSS API plug-in name must be separated by a comma (,) with no space either before or after the comma. Plug-in names should be listed in the order of preference.

srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server

This parameter specifies the name of the default userid-password plug-in library to be used for server-side authentication.

It handles incoming connections at the server when the **srvcon_auth** parameter is specified as CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT or DATA_ENCRYPT_CMP, or when **srvcon_auth** is not specified, and **authentication** is specified as CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT or DATA_ENCRYPT_CMP.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [any valid string]

By default, the value is null and the userid-password plug-in library that is supplied with DB2 database is used. For non-root installations, if the DB2 userid and password plug-in library is used, the **db2rfe** command must be run before using your DB2 database product.

srv_plugin_mode - Server plug-in mode

This parameter specifies whether plug-ins are to run in fenced mode or unfenced mode. Unfenced mode is the only supported mode.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
UNFENCED

ssl_cipherspecs - Supported cipher specifications at the server configuration parameter

This configuration parameter specifies the cipher suites that the server allows for incoming connection requests when using SSL protocol.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [`TLS_RSA_WITH_AES_256_CBC_SHA`; `TLS_RSA_WITH_AES_128_CBC_SHA` ; `TLS_RSA_WITH_3DES_EDE_CBC_SHA`]

You can specify multiple cipher specifications, such as `TLS_RSA_WITH_AES_256_CBC_SHA` or `TLS_RSA_WITH_AES_128_CBC_SHA` or `TLS_RSA_WITH_3DES_EDE_CBC_SHA`. They must be separated by a comma (,) with no space either before or after the comma.

During SSL handshake, if null or multiple values are specified, the client and the server negotiate and find the most secure cipher suites to use. If no compatible cipher suites is found, the connection fails. You cannot prioritize the cipher suites by specifying one before the another.

ssl_clnt_keydb - SSL key file path for outbound SSL connections at the client configuration parameter

This configuration parameter specifies the key file to be used for SSL connection at the client-side.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [any valid path; GSK_MS_CERTIFICATE_STORE]

This parameter specifies a fully qualified file path for the key file; or, on Windows only, the keyword GSK_MS_CERTIFICATE_STORE to use the Microsoft Windows Certificate Store.

The SSL key file has extension `.kbd` by default, and stores the signer certificate from the servers personal certificate. For a self-signed server personal certificate, the signer certificate is the public key. For a certificate authority signed server personal certificate, the signer certificate is the root CA certificate. The key file is accessed by the client to verify the servers personal certificate during the SSL handshake.

By default, the value is null. Depending on your application type, you should specify the client SSL key file path by the database manager configuration parameter **ssl_clnt_keydb**, the connection string **ssl_clnt_keydb**, or the `db2cli.ini` and `db2dsdriver.cfg` keyword **SSLClientKeystoredb** for a SSL connection request. If none of them is specified, the SSL connection fails.

ssl_clnt_stash - SSL stash file path for outbound SSL connections at the client configuration parameter

This configuration parameter specifies the fully qualified file path of the stash file to be used for SSL connections at the client-side.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [any valid path]

The SSL stash file has extension `.sth` by default, and stores an encrypted version of the key database password. The password held in the stash file is used to access the SSL key file during an SSL connection request.

On Windows platforms, **ssl_clnt_stash** is not required if **ssl_clnt_keydb** is set to the keyword GSK_MS_CERTIFICATE_STORE.

By default the value is null. Depending on your application type, you can specify the client SSL stash file path by the database manager configuration parameter **ssl_clnt_stash**, the connection string **ssl_clnt_stash**, or the `db2cli.ini` keyword **ssl_clnt_stash** for a SSL connection request. If none of them is specified, the SSL connection fails.

ssl_svr_keydb - SSL key file path for incoming SSL connections at the server configuration parameter

This configuration parameter specifies the key file to be used for SSL setup at server-side.

Configuration type
Database manager

Applies to

- Database server with local and remote clients

- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [any valid path; GSK_MS_CERTIFICATE_STORE]

This parameter specifies a fully qualified file path of the key file or on Windows only, the keyword GSK_MS_CERTIFICATE_STORE which indicates to use Microsoft Windows Certificate Store.

The SSL key file has extension `.kdb` by default, and stores personal certificates, personal certificate requests and signer certificates. This key file is accessed during the instance startup and the servers personal certificate is sent to the client for server authentication during SSL handshake.

By default, the value is null. During the instance start up, you must define if the **DB2COMM** registry variable contains `SSL`. Otherwise, the instance starts up without SSL protocol support.

ssl_svr_label - Label in the key file for incoming SSL connections at the server configuration parameter

This configuration parameter specifies a label of the personal certificate of the server in the key database.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default
Null

By default, the value is null. When establishing a SSL connection, the server certificate specified by this configuration parameter is sent to the client for server authentication. If the value is null, the default certificate defined in the key file is used. If the default does not exist, the connection fails.

ssl_svr_stash - SSL stash file path for incoming SSL connections at the server configuration parameter

This configuration parameter specifies a fully qualified file path of the stash file to be used for SSL setup at server-side.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [any valid path]

The SSL stash file has extension `.sth` by default, and stores an encrypted version of the key database password. The password held in the stash file is used to access the SSL key file during the instance startup.

By default, the value is null. During the instance start up, you must define if the **DB2COMM** registry variable contains `SSL`. Otherwise, the instance starts up without SSL protocol support.

On Windows platforms, **ssl_svr_stash** is not required if **ssl_svr_keydb** is set to the keyword `GSK_MS_CERTIFICATE_STORE`.

start_stop_time - Start and stop timeout

This parameter specifies the time, in minutes, within which all database partition servers must respond to a **START DBM** or a **STOP DBM** command. It is also used as the timeout value during **ADD DBPARTITIONNUM** and **DROP DBPARTITIONNUM** operations.

Configuration type
Database manager

Applies to
Database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
`10 [1 - 1 440]`

Unit of measure
Minutes

Member or nodes that do not respond to **db2start** or **db2stop** commands within the specified time will be killed and cleaned up automatically by **db2start** or **db2stop** in a multi member/node instance. The diagnostic messages are logged into the **diagpath** defined in the database manager configuration or at its default value (for example, `sqllib/db2dump/ $m` on UNIX operating systems).

If a **db2start** or **db2stop** operation in a multi-partition database is not completed within the value specified by the **start_stop_time** database manager configuration parameter, the database partitions that have timed out will be killed and cleaned up automatically. Environments with many database partitions with a low value for **start_stop_time** might experience this behavior. To resolve this behavior, increase the value of **start_stop_time**.

When adding a new database partition using one of the **db2start**, **START DATABASE MANAGER**, or **ADD DBPARTITIONNUM** commands, the add database partition operation must determine whether or not each database in the instance is enabled for automatic storage. This is done by communicating with the catalog partition for

each database. If automatic storage is enabled, the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partitions, the operation might have to communicate with another database partition server to retrieve the table space definitions for the database partitions that reside on that server. These factors should be considered when determining the value of the **start_stop_time** parameter.

ssl_svcename - SSL service name configuration parameter

This configuration parameter specifies the name of the port that a database server uses to await communications from remote client nodes using SSL protocol.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default
Null

This configuration parameter contains the port that a database server uses to await communications from remote client nodes through SSL protocol. This service name must be reserved for use by the database manager. During instance startup, you must define if the **DB2COMM** registry variable contains `SSL`. Otherwise the instance starts up without SSL protocol support.

If **DB2COMM** contains both `TCPIP` and `SSL`, the port specified by **ssl_svcename** must not be the same as the **svcename**. Otherwise, the instance starts up without either SSL or TCP/IP protocol support.

On UNIX operating systems, the `services` file is located in: `/etc/services`

The database server SSL port (number *n*) and its service name needs to be defined in the services file on the database client.

ssl_versions - Supported SSL versions at the server configuration parameter

This configuration parameter specifies Secure Sockets Layer (SSL) and Transport Layer Security (TLS) versions that the server supports for incoming connection requests.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [TLSv1]

If you set the parameter to null or TLSv1, the parameter enables support for TLS version 1.0 (RFC2246) and TLS version 1.1 (RFC4346).

During SSL handshake, the client and the server negotiate and find the most secure version to use either TLS version 1.0 or TLS version 1.1. If there is no compatible version between the client and the server, the connection fails. If the client supports TLS version 1.0 and TLS version 1.1, but the server support TLS version 1.0 only, then TLS version 1.0 is used.

svcename - TCP/IP service name

This parameter contains the name of the TCP/IP port which a database server will use to await communications from remote client nodes. This name must be the reserved for use by the database manager.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default
Null

In order to accept connection requests from a Data Server Runtime Client using TCP/IP, the database server must be listening on a port designated to that server. The system administrator for the database server must reserve a port (number *n*) and define its associated TCP/IP service name in the services file at the server.

The database server port (number *n*) and its TCP/IP service name need to be defined in the services file on the database client.

On Linux and UNIX systems, the services file is located in: /etc/services

The **svcename** parameter should be set to the port number or the service name associated with the main connection port so that when the database server is started, it can determine on which port to listen for incoming connection requests.

sysadm_group - System administration authority group name

This parameter defines the group name with SYSADM authority for the database manager instance.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter type
Configurable

Default
NULL

The SYSADM authority level is the highest level of administrative authority at the instance level. Users with SYSADM authority can run some utilities and issue some database and database manager commands within the instance.

SYSADM authority is determined by the security facilities used in a specific operating environment.

- For the Windows operating system, this parameter can be set to local or domain group. Group names must adhere to the length limits specified in SQL and XML limits. The following users have SYSADM authority if "NULL" is specified for **sysadm_group** database manager configuration parameter:
  - Members of the local Administrators group
  - Members of the Administrators group at the Domain Controller if **DB2_GRP_LOOKUP** is not set or set to DOMAIN
  - Members of DB2ADMNS group if Extended Security feature is enabled. The location of the DB2ADMNS group was decided during installation
  - The LocalSystem account
- For Linux and UNIX systems, if NULL is specified as the value of this parameter, the SYSADM group defaults to the primary group of the instance owner.

  If the value is not NULL, the SYSADM group can be any valid UNIX group name.

To restore the parameter to its default (NULL) value, use **UPDATE DBM CFG USING SYSADM_GROUP NULL**. You must specify the keyword NULL in uppercase.

sysctrl_group - System control authority group name

This parameter defines the group name with system control (SYSCTRL) authority. SYSCTRL has privileges allowing operations affecting system resources, but does not allow direct access to data.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default
Null

Group names on all platforms are accepted as long as they adhere to the length limits specified in SQL and XML limits.

sysmaint_group - System maintenance authority group name

This parameter defines the group name with system maintenance (SYSMAINT) authority.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default
Null

SYSMAINT has privileges to perform maintenance operations on all databases associated with an instance without having direct access to data.

Group names on all platforms are accepted as long as they adhere to the length limits specified in SQL and XML limits.

To restore the parameter to its default (`NULL`) value, use **`UPDATE DBM CFG USING SYSMAINT_GROUP NULL`**. You must specify the keyword `NULL` in uppercase.

sysmon_group - System monitor authority group name

This parameter defines the group name with system monitor (SYSMON) authority.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default
`Null`

Users having SYSMON authority at the instance level have the ability to take database system monitor snapshots of a database manager instance or its databases. Refer to System monitor authority (SYSMON) for a full list of commands supported by SYSMON authority.

Users with SYSADM, SYSCTRL, or SYSMAINT authority automatically have the ability to take database system monitor snapshots and to use these commands.

Group names on all platforms are accepted as long as they adhere to the length limits specified in "SQL and XML limits" in the *Database Administration Concepts and Configuration Reference*.

To restore the parameter to its default (NULL) value, use **UPDATE DBM CFG USING SYSMON_GROUP NULL**. You must specify the keyword NULL in uppercase.

tm_database - Transaction manager database name

This parameter identifies the name of the transaction manager (TM) database for each DB2 instance.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
1ST_CONN [any valid database name]

A TM database can be:

- A local DB2 database
- A remote DB2 database that does not reside on a host or AS/400 system
- A DB2 for OS/390 Version 5 database if accessed via TCP/IP and the sync point manager (SPM) is not used.

The TM database is a database that is used as a logger and coordinator, and is used to perform recovery for indoubt transactions.

You can set this parameter to **1ST_CONN**, which will set the TM database to be the first database to which a user connects.

Recommendation: For simplified administration and operation, you might want to create a few databases over a number of instances and use these databases exclusively as TM databases.

tp_mon_name - Transaction processor monitor name

This parameter identifies the name of the transaction processing (TP) monitor product being used.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default
No default

Valid values

- CICS®
- MQ
- CB
- SF
- TUXEDO
- TOPEND
- blank or some other value (for UNIX and Windows; no other possible values for Solaris or SINIX)

- If applications are run in a WebSphere Enterprise Server Edition CICS environment, this parameter should be set to “CICS”
- If applications are run in a WebSphere Enterprise Server Edition Component Broker environment, this parameter should be set to “CB”
- If applications are run in an IBM MQSeries® environment, this parameter should be set to “MQ”
- If applications are run in a BEA Tuxedo environment, this parameter should be set to “TUXEDO”
- If applications are run in an IBM San Francisco environment, this parameter should be set to “SF”.

IBM WebSphere EJB and **Microsoft Transaction Server** users do not need to configure any value for this parameter.

If none of the products mentioned previously are being used, this parameter should not be configured but left blank.

In previous versions of IBM DB2 on Windows, this parameter contained the path and name of the DLL which contained the XA Transaction Manager's functions *ax_reg* and *ax_unreg*. This format is still supported. If the value of this parameter does not match any of the previously mentioned TP Monitor names, it will be assumed that the value is a library name which contains the *ax_reg* and *ax_unreg* functions. This is true for UNIX and Windows environments.

TXSeries CICS Users: In previous versions of this product on Windows it was required to configure this parameter as “libEncServer:C” or “libEncServer:E”. While this is still supported, it is no longer required. Configuring the parameter as “CICS” is sufficient.

MQSeries Users: In previous versions of this product on Windows it was required to configure this parameter as “mqmax”. While this is still supported, it is no longer required. Configuring the parameter as “MQ” is sufficient.

Component Broker Users: In previous versions of this product on Windows it was required to configure this parameter as “somtrx1i”. While this is still supported, it is no longer required. Configuring the parameter as “CB” is sufficient.

San Francisco Users: In previous versions of this product on Windows it was required to configure this parameter as “ibmsfDB2”. While this is still supported, it is no longer required. Configuring the parameter as “SF” is sufficient.

The maximum length of the string that can be specified for this parameter is 19 characters.

It is also possible to configure this information in IBM DB2 Version 9.1's XA OPEN string. If multiple Transaction Processing Monitors are using a single DB2 instance, then it will be required to use this capability.

trust_allclnts - Trust all clients

This parameter and **trust_clntauth** are used to determine where users are validated to the database environment.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
YES [NO, YES, DRDAONLY]

This parameter is only active when the **authentication** parameter is set to CLIENT.

By accepting the default of YES for this parameter, all clients are treated as trusted clients. This means that the server assumes that a level of security is available at the client and the possibility that users can be validated at the client.

This parameter can only be changed to NO if the **authentication** parameter is set to CLIENT. If this parameter is set to NO, the untrusted clients must provide a userid and password combination when they connect to the server. Untrusted clients are operating system platforms that do not have a security subsystem for authenticating users.

Setting this parameter to DRDAONLY protects against all clients except clients from DB2 for OS/390 and z/OS, DB2 for VM and VSE, and DB2 for OS/400®. 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.

When **trust_allclnts** is set to DRDAONLY, the **trust_clntauth** parameter is used to determine where the clients are authenticated. If **trust_clntauth** is set to CLIENT, authentication occurs at the client. If **trust_clntauth** is set to SERVER, authentication occurs at the client if no password is provided, and at the server if a password is provided.

trust_clntauth - Trusted clients authentication

This parameter specifies whether a trusted client is authenticated at the server or the client when the client provides a userid and password combination for a connection.

This parameter (and **trust_allclnts**) is only active if the **authentication** parameter is set to CLIENT. If a user ID and password are not provided, the client is assumed to have validated the user, and no further validation is performed at the server.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
`CLIENT` [`CLIENT`, `SERVER`]

If this parameter is set to `CLIENT` (the default), the trusted client can connect without providing a user ID and password combination, and the assumption is that the operating system has already authenticated the user. If it is set to `SERVER`, the user ID and password will be validated at the server.

When using the db2CfgSet API to set the database manager configuration parameter, the numeric value for CLIENT is 0. The numeric value for SERVER is 1.

util_impact_lim - Instance impact policy

This parameter allows the database administrator (DBA) to limit the performance degradation of a throttled utility on the workload.

Configuration type
Database manager

Applies to

- Database server with local clients
- Database server with local and remote clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
`10` [`1 - 100`]

Unit of measure
Percentage of allowable impact on workload

If the performance degradation is limited, the DBA can then run online utilities during critical production periods, and be guaranteed that the performance impact on production work will be within acceptable limits.

For example, a DBA specifying a **`util_impact_lim`** (impact policy) value of `10` can expect that a throttled backup invocation will not impact the workload by more than 10 percent.

If **`util_impact_lim`** is `100`, no utility invocations will be throttled. In this case, the utilities can have an arbitrary (and undesirable) impact on the workload. If **`util_impact_lim`** is set to a value that is less than `100`, it is possible to invoke utilities in throttled mode. To run in throttled mode, a utility must also be invoked with a non-zero priority.

Recommendation: Most users will benefit from setting **util_impact_lim** to a low value (for example, between 1 and 10).

A throttled utility will usually take longer to complete than an unthrottled utility. If you find that a utility is running for an excessively long time, increase the value of **util_impact_lim**, or disable throttling altogether by setting **util_impact_lim** to 100.

wlm_dispatcher - Workload management dispatcher

This parameter enables (YES) or disables (NO) the DB2 workload management dispatcher. By default, an enabled dispatcher allows the setting of the CPU limits.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
NO [NO; YES]

When upgrading the DB2 database manager, the value of the **wlm_dispatcher** database manager configuration parameter is set to NO.

The workload management dispatcher provides CPU scheduling capabilities at the service class level in the DB2 database manager using shares-based allocation of CPU resources, or CPU limits, or both.

With the workload management dispatcher enabled, all work running in user and maintenance service classes is placed under the control of the dispatcher. When enabled, CPU limit settings are enforced by the dispatcher as the default case. In order to use shares-based allocation of CPU resources, the **wlm_disp_cpu_shares** database manager configuration parameter must be enabled.

When the **wlm_dispatcher** configuration parameter is set to YES, the following conditions apply:

- If any service class has an agent priority set to any value other than the default, a warning message is written to the db2diag log and the administration notification log at the time of database activation.
- Any attempt to create or alter a service class to set agent priority to a value other than the default value results in a warning being returned to the application that issued the statement to create or alter the service class.

wlm_disp_concur - Workload manager dispatcher thread concurrency

This parameter specifies how the DB2 workload manager (WLM) dispatcher sets the thread concurrency level. You can also manually set the thread concurrency level to a fixed value.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
COMPUTED [COMPUTED; *manually_set_value*]

When upgrading DB2 database manager, the value of the **wlm_disp_concur** database manager configuration parameter is COMPUTED.

COMPUTED
DB2 database manager computes a fixed thread concurrency level based upon the value of 4 times the number of logical CPUs available to the DB2 database manager.

manually_set_value
You can manually set the thread concurrency level to a fixed value (1 - 32767). The optimal value depends on the specific hardware used and the operating system level; generally, in the range of 2 to 4 times the number of logical CPUs on the host or LPAR.

Unit of measure
Number of concurrent threads

The setting of this database manager configuration parameter controls the number of threads that the WLM dispatcher allows to be dispatched to the operating system run queues in parallel. The value is set as a low multiple of the number of logical CPUs available to the DB2 database manager. In general, you can set the value to 4 times the number of available logical CPUs to take into account possible scheduling latencies that result when threads switch in and out of the active state. An optimal value is just large enough to ensure that there are an adequate numbers of threads for the DB2 database manager to fully use the CPUs on the host or LPAR and no larger. This optimal value ensures maximum efficiency and gives the DB2 WLM dispatcher maximum control over CPU allocation.

wlm_disp_cpu_shares - Workload manager dispatcher CPU shares

This parameter enables (YES) or disables (NO) the control of CPU shares by the DB2 workload manager (WLM) dispatcher. By default, an enabled WLM dispatcher controls only CPU limits.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
NO [NO; YES]

When upgrading DB2 database manager, the value of the **wlm_disp_cpu_shares** database manager configuration parameter is NO.

If the value of the **wlm_dispatcher** database manager configuration parameter is set to YES and the value of the **wlm_disp_cpu_shares** database manager configuration parameter is set to NO, the WLM dispatcher can apply only CPU limits to the management of service classes.

If the value of the **wlm_dispatcher** database manager configuration parameter is set to YES and the value of the **wlm_disp_cpu_shares** database manager configuration parameter is set to YES, the WLM dispatcher can apply both CPU limits and CPU shares to the management of service classes. By default, all service classes are assigned 1000 hard CPU shares to ensure an equal division of CPU resources.

Table 135. Summary of required database manager configuration parameter settings for service class management by the DB2 WLM dispatcher

Service class management	Setting of wlm_dispatcher	Setting of wlm_disp_cpu_shares
None	NO	NO
CPU limits	YES	NO
CPU limits + CPU shares	YES	YES

wlm_disp_min_util - Workload manager dispatcher minimum CPU utilization

This parameter specifies the minimum amount of CPU utilization that is necessary for a service class to be included in the DB2 WLM-managed sharing of CPU resources.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Multimember database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
5 [0 to 100]

When upgrading DB2 database manager, the value of the **wlm_disp_min_util** database manager configuration parameter is 5.

Unit of measure
Percent

To illustrate the usage of this database manager configuration parameter with an example, suppose there are three service classes, A, B, and C, and each has 1000 shares of the CPU resources. In this example, the same result is obtained whether the service class shares are hard or soft CPU shares. Service classes A and B each have a CPU utilization that is greater than or equal to the 8% value set for the **wlm_disp_min_util** configuration parameter. Service class C has a 3% CPU utilization that is less than the 8% value set for the **wlm_disp_min_util** configuration parameter. In CPU share calculations, service class C is considered to not have any executing work. Therefore, only service classes A and B equally share the CPU resources and each receives a 50% share. When service class C begins to execute work to an extent that the CPU utilization is greater than or equal to the 8% value set for the **wlm_disp_min_util** configuration parameter, at this point service classes A, B, and C are now considered to equally share the CPU resources and each receives a 33.3% share.

In multimember database environments, it is the aggregate of the CPU utilizations of all the members on a host or LPAR that is compared to the **wlm_disp_min_util** configuration parameter to determine if the host or LPAR is included in the WLM-managed sharing of CPU resources.

Database configuration parameters

alt_collate - Alternate collating sequence

This parameter specifies the collating sequence that is to be used for Unicode tables in a non-Unicode database.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
Null [IDENTITY_16BIT]

Until this parameter is set, Unicode tables and routines cannot be created in a non-Unicode database. Once set, this parameter cannot be changed or reset.

This parameter cannot be set for Unicode databases.

app_ctl_heap_sz - Application control heap size

This parameter has been deprecated since Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the database manager in DB2 Version 9.5 or later releases. In Version 9.5, it has been replaced by the **appl_memory** configuration parameter.

Note: The following information applies only to pre-Version 9.5 data servers and clients.

For partitioned databases, and for non-partitioned databases with intra-parallelism enabled (**intra_parallel**=ON), this parameter specifies the average size of the shared memory area allocated for an application. For non-partitioned databases where intra-parallelism is disabled (**intra_parallel**=OFF), this is the maximum private memory that will be allocated for the heap. There is one application control heap per connection per database partition.

Configuration type
Database

Parameter type
Configurable

Default [range]

Database server with local and remote clients

- 128 [1 - 64 000] when **intra_parallel** is not enabled
- 512 [1 - 64 000] when **intra_parallel** is enabled

Database server with local clients

- 64 [1 - 64 000] (for non-UNIX platforms) when **intra_parallel** is not enabled
- 512 [1 - 64 000] (for non-UNIX platforms) when **intra_parallel** is enabled
- 128 [1 - 64 000] (for Linux and UNIX platforms) when **intra_parallel** is not enabled
- 512 [1 - 64 000] (for Linux and UNIX platforms) when **intra_parallel** is enabled

Partitioned database server with local and remote clients
512 [1 - 64 000]

Unit of measure
Pages (4 KB)

When allocated
When an application starts

When freed
When an application completes

The application control heap is required primarily for sharing information between agents working on behalf of the same request. Usage of this heap is minimal for non-partitioned databases when running queries with a degree of parallelism equal to 1.

This heap is also used to store descriptor information for declared temporary tables. The descriptor information for all declared temporary tables that have not been explicitly dropped is kept in this heap's memory and cannot be dropped until the declared temporary table is dropped.

Recommendation: Initially, start with the default value. You might have to set the value higher if you are running complex applications, if you have a system that contains a large number of database partitions, or if you use declared temporary tables. The amount of memory needed increases with the number of concurrently active declared temporary tables. A declared temporary table with many columns

has a larger table descriptor size than a table with few columns, so having a large number of columns in an application's declared temporary tables also increases the demand on the application control heap.

appgroup_mem_sz - Maximum size of application group memory set

This parameter has been deprecated since Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the database manager in DB2 Version 9.5 or later releases. In Version 9.5, it has been replaced by the **appl_memory** configuration parameter.

Note: The following information applies only to pre-Version 9.5 data servers and clients.

This parameter determines the size of the application group shared memory segment.

Configuration type
Database

Parameter type
Configurable

Default [range]

UNIX Database server with local clients (other than 32-bit HP-UX)
`20 000 [1 - 1 000 000 ]`

32-bit HP-UX

- Database server with local clients
- Database server with local and remote clients
- Partitioned database server with local and remote clients

`10 000 [1 - 1 000 000 ]`

Windows Database server with local clients
`10 000 [1 - 1 000 000 ]`

Database server with local and remote clients (other than 32-bit HP-UX)
`30 000 [1 - 1 000 000 ]`

Partitioned database server with local and remote clients (other than 32-bit HP-UX)
`40 000 [1 - 1 000 000 ]`

Unit of measure
Pages (4 KB)

Information that needs to be shared between agents working on the same application is stored in the application group shared memory segment.

In a partitioned database, or in a non-partitioned database with intrapartition parallelism enabled or concentrator enabled, multiple applications share one application group. One application group shared memory segment is allocated for the application group. Within the application group shared memory segment, each application will have its own application control heap, and all applications will share one application group shared heap.

The number of applications in one application group is calculated by:

```
appgroup_mem_sz / app_ctl_heap_sz
```

The application group shared heap size is calculated by:

```
appgroup_mem_sz * groupheap_ratio / 100
```

The size of each application control heap is calculated by:

```
app_ctl_heap_sz * (100 - groupheap_ratio) / 100
```

Recommendation: Retain the default value of this parameter unless you are experiencing performance problems.

appl_memory - Application Memory configuration parameter

This parameter allows DBAs and ISVs to control the maximum amount of application memory that is allocated by DB2 database agents to service application requests.

By default, its value is set to AUTOMATIC, meaning that all application memory requests will be allowed as long as the total amount of memory allocated by the database partition is within the **instance_memory** limits.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Default [range]
Automatic [128 - 4 294 967 295]

Unit of measure
Pages (4 KB)

When allocated
During database activation

When freed
During database deactivation

Note: When **appl_memory** is set to AUTOMATIC, the initial application memory allocation at database activation time is minimal, and increases (or decreases) as needed. The change is applied in memory and the value of **appl_memory** does not change on disk as shown by **db2 get db cfg show detail**. On next activation, the value will be recalculated. If **appl_memory** is set to a specific value, then the requested amount of memory is allocated initially during database activation, and the application memory size does not change. If the initial amount of application memory cannot be allocated from the operating system, or exceeds the **instance_memory** limit, database activation fails with an SQL1084C error (Shared memory segments cannot be allocated).

applheapsz - Application heap size

The **applheapsz** configuration parameter refers to the total amount of application memory that can be consumed by the entire application.

In versions of DB2 prior to Version 9.5, the **applheapsz** database configuration parameter referred to the amount of application memory each individual database agent working for that application could consume.

With Version 9.5, this database configuration parameter has a default value of AUTOMATIC, meaning that it increases as needed until either the **appl_memory** limit is reached, or the **instance_memory** limit is reached. For partitioned database environments, Concentrator, or SMP configurations, this means that the **applheapsz** value used in previous releases may need to be increased under similar workloads, unless the AUTOMATIC setting is used.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Default [range]
Automatic [16 - 60 000]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
When an application associates with, or connects to, a database.

When freed
When the application disassociates or disconnects from the database.

Note: This parameter defines the maximum size of the application heap. One application heap is allocated per database application when the application first connects with the database. The heap is shared by all database agents working for that application. (In previous releases, each database agent allocated its own application heap.) Memory is allocated from the application heap as needed to process the application, up to the limit specified by this parameter. When set to AUTOMATIC, the application heap is allowed to grow as needed up to either the **appl_memory** limit for the database, or the **instance_memory** limit for the database partition. The entire application heap is freed when the application disconnects with the database.

The online changed value takes effect at an application connection boundary, that is, after it is changed dynamically, currently connected applications still use the old value, but all newly connected applications will use the new value.

archretrydelay - Archive retry delay on error

This parameter specifies the number of seconds to wait after a failed archive attempt before trying to archive the log file again.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Default [range]
20 [0 - 65 535]

Subsequent retries will only take affect if the value of the **numarchretry** database configuration parameter is at least 1.

auto_del_rec_obj - Automated deletion of recovery objects configuration parameter

This parameter specifies whether database log files, backup images, and load copy images should be deleted when their associated recovery history file entry is pruned.

Configuration type
Database

Parameter type
Configurable online

Propagation class
Immediate

Default [range]
OFF [ON; OFF]

You can prune the entries in the recovery history file using the **PRUNE HISTORY** command or the db2Prune API. You can also configure the IBM Data Server database manager to automatically prune the recovery history file after each full database backup. If you set the **auto_del_rec_obj** database configuration parameter to ON, then the database manager will also delete the corresponding physical log files, backup images, and load copy images when it prunes the history file. The database manager can only delete recovery objects such as database logs, backup images, and load copy images when your storage media is disk, or if you are using a storage manager, such as the Tivoli Storage Manager. If the **logarchmeth1** parameter is set to LOGRETAIN and the **ARCHIVE LOG** command is issued, the log files will not be deleted by the prune even if entries appear in the history file and **auto_del_rec_obj** is set to ON.

auto_maint - Automatic maintenance

This parameter is the parent of all the other automatic maintenance database configuration parameters (**auto_db_backup**, **auto_tbl_maint**, **auto_runstats**, **auto_stats_prof**, **auto_stmt_stats**, **auto_stats_views**, **auto_prof_upd**, **auto_reorg**, and **auto_sampling**).

Configuration type
Database

Applies to

- Database server with local and remote clients

- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
ON [ON; OFF]

When this parameter is disabled, all of its child parameters are also disabled, but their settings, as recorded in the database configuration file, do not change. When this parent parameter is enabled, recorded values for its child parameters take effect. In this way, automatic maintenance can be enabled or disabled globally.

By default, this parameter is set to ON.

You can enable or disable individual automatic maintenance features independently by setting the following parameters:

auto_db_backup
This automated maintenance parameter enables or disables automatic backup operations for a database. A backup policy (a defined set of rules or guidelines) can be used to specify the automated behavior. The objective of the backup policy is to ensure that the database is being backed up regularly. The backup policy for a database is created automatically when the DB2 Health Monitor first runs. By default, this parameter is set to OFF. To be enabled, this parameter must be set to ON, and its parent parameter must also be enabled.

auto_tbl_maint
This parameter is the parent of all table maintenance parameters (**auto_runstats**, **auto_stats_prof**, **auto_prof_upd**, and **auto_reorg**). When this parameter is disabled, all of its child parameters are also disabled, but their settings, as recorded in the database configuration file, do not change. When this parent parameter is enabled, recorded values for its child parameters take effect. In this way, table maintenance can be enabled or disabled globally.

By default, this parameter is set to ON.

auto_runstats
This automated table maintenance parameter enables or disables automatic table **RUNSTATS** operations for a database. A **RUNSTATS** policy (a defined set of rules or guidelines) can be used to specify the automated behavior. Statistics collected by the **RUNSTATS** utility are used by the optimizer to determine the most efficient plan for accessing the physical data. To be enabled, this parameter must be set to ON, and its parent parameters must also be enabled.

By default, this parameter is set to ON.

auto_stats_prof

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*.

When enabled, this automated table maintenance parameter turns on statistical profile generation, designed to improve applications whose workloads include complex queries, many predicates, joins, and grouping operations over several tables. To be enabled, this parameter must be set to ON, and its parent parameters must also be enabled.

By default, this parameter is set to OFF.

Automatic statistics profile generation is not supported in partitioned database environments, in certain federated database environments, in DB2 pureScale environments, or when intrapartition parallelism is enabled. Automatic statistics profile generation cannot be enabled if the **section_actuals** database configuration parameter is enabled (SQLCODE -5153).

auto_stmt_stats

This parameter enables and disables the collection of real-time statistics. It is a child of the **auto_runstats** configuration parameter. This feature is enabled only if the parent, **auto_runstats** configuration parameter, is also enabled. For example, to enable **auto_stmt_stats**, set **auto_maint**, **auto_tbl_maint**, and **auto_runstats** to ON. To preserve the child value, the **auto_runstats** configuration parameter can be ON while the **auto_maint** configuration parameter is OFF. The corresponding Auto Runstats feature will still be OFF.

Assuming that both Auto Runstats and Auto Reorg are enabled, the settings are as follows:

```
    Automatic maintenance                        (AUTO_MAINT) = ON
      Automatic database backup                (AUTO_DB_BACKUP) = OFF
        Automatic table maintenance            (AUTO_TBL_MAINT) = ON
          Automatic runstats                    (AUTO_RUNSTATS) = ON
             Real-time statistics              (AUTO_STMT_STATS) = ON
             Statistical views                (AUTO_STATS_VIEWS) = OFF
             Automatic sampling                  (AUTO_SAMPLING) = OFF
   Automatic statistics profiling      (AUTO_STATS_PROF) = OFF
             Statistics profile updates          (AUTO_PROF_UPD) = OFF
          Automatic reorganization                (AUTO_REORG) = ON
```

You can disable both Auto Runstats and Auto Reorg features temporarily by setting **auto_tbl_maint** to OFF. Both features can be enabled later by setting **auto_tbl_maint** back to ON. You do not need to issue **db2stop** or **db2start** commands to have the changes take effect.

By default, this parameter is set to ON.

auto_stats_views

This parameter enables and disables automatic statistic collection on statistical views. The statistics on statistical views are automatically maintained when this parameter is set to ON.

By default, this parameter is set to OFF.

auto_prof_upd

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*.

When enabled, this automated table maintenance parameter (a child of **auto_stats_prof**) specifies that the **RUNSTATS** profile is to be updated with recommendations. When this parameter is disabled, recommendations are stored in the opt_feedback_ranking table, which you can inspect when manually updating the **RUNSTATS** profile. To be enabled, this parameter must be set to ON, and its parent parameters must also be enabled.

By default, this parameter is set to OFF.

auto_reorg
This automated table maintenance parameter enables or disables automatic table and index reorganization for a database. A reorganization policy (a defined set of rules or guidelines) can be used to specify the automated behavior. To be enabled, this parameter must be set to ON, and its parent parameters must also be enabled.

By default, this parameter is set to OFF.

auto_sampling
This parameter controls whether automatic statistics collection uses sampling when collecting statistics for a large table. To enable the automatic sampling, set **auto_maint**, **auto_tbl_maint**, **auto_runstats** and **auto_sampling** to ON. If the automatic statistics collection is enabled, page-level sampling is used and the sampling rate is determined automatically. Both data and index pages are sampled for base tables as opposed to statistical views, which do not have indexes.

By default, this parameter is set to OFF.

auto_reval - Automatic revalidation and invalidation configuration parameter

This configuration parameter controls the revalidation and invalidation semantics.

Configuration type
Database

Parameter type
- Configurable online
- Configurable by member in a DB2 pureScale environment

Default [range]
DEFERRED [IMMEDIATE, DISABLED, DEFERRED, DEFERRED_FORCE]

If you create a new database, by default this configuration parameter is set to DEFERRED.

If you upgrade a database from Version 9.5, or earlier, **auto_reval** is set to DISABLED. The revalidation behavior is the same as in the previous releases.

If you set this parameter to IMMEDIATE it means that all dependent objects will be revalidated immediately after objects are invalidated. This applies to some DDL statements, such as ALTER TABLE, ALTER COLUMN, or CREATE OR REPLACE. The successful revalidation of the dependent objects does not rely on any other DDL changes; therefore, revalidation can be completed immediately.

If you set this parameter to DEFERRED, it means that all dependent objects are revalidated the next time that they are accessed.

Note that if you set this parameter either to IMMEDIATE or DEFERRED, and if any revalidation operation fails, the invalid dependent objects will remain invalid until the next time that they are accessed.

If you set this parameter to DEFERRED_FORCE it behaves the same way as when it is set to DEFERRED and an additional CREATE with error feature is enabled.

In some cases, the syntax that you explicitly specify might override the setting of **auto_reval**. For example, if you use the DROP COLUMN clause of the ALTER TABLE statement without specifying CASCADE or RESTRICT, the semantics are controlled by **auto_reval**. However, if you specify CASCADE or RESTRICT, the previous cascade or restrict semantics are used, overriding the new semantics specified by **auto_reval**.

This configuration parameter is dynamic, meaning that a change in its value takes effect immediately. You do not have to reconnect to the database for the change to take effect.

autorestart - Auto restart enable

The **autorestart** parameter determines whether the database manager will automatically initiate crash recovery when a user connects to a database that had previously terminated abnormally. If the **autorestart** parameter is not set, the user will be required to issue an explicit restart database command before being able to connect to the database.

Configuration type
Database

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
On [On; Off]

When the **autorestart** parameter is set to ON, the next database connection attempt performs crash recovery automatically if necessary.

In a DB2 pureScale environment, the automatic restart behavior changes. If a database member fails, member crash recovery is automatically initiated for that member. In addition, if both cluster caching facilities fail at the same time, group crash recovery is automatically initiated. During member crash recovery, no connections are allowed against the database through that member. An SQL1015N error is returned if you try to connect to the database through a member that requires crash recovery. If crash recovery fails, another attempt at crash recovery is made. If the second attempt at crash recovery fails, an ALERT is set for the member on the host where recovery fails and restart light for that member is performed on a different host.

When the **autorestart** parameter is set to OFF, the automated member crash recovery and group crash recovery processes are inactive and must be performed manually, if required. The crash recovery processes can be initiated with the **RESTART DATABASE** command.

In a DB2 pureScale environment, if a member fails and cannot be restarted on its original host (also known as home host), DB2 cluster services restarts that member on one of the other available member hosts in the DB2 pureScale cluster. This is known as restart light processing. When a member is in restart light mode, you cannot directly connect to it, therefore no manual restart is possible. You must first set the **autorestart** parameter back to ON. The restart light member then automatically detects the change in the database configuration parameter and initiates crash recovery if needed.

avg_appls - Average number of active applications

This parameter is used by the query optimizer to help estimate how much buffer pool space will be available at run time for the access plan chosen.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Statement boundary

Default [range]
Automatic [1 - maxappls]

Unit of measure
Counter

Recommendation: When running the DB2 database product in a multi-user environment, particularly with complex queries and a large buffer pool, you might want the query optimizer to know that multiple query users are using your system so that the optimizer should be more conservative in assumptions of buffer pool availability.

When this parameter is set to AUTOMATIC, the parameter is updated to an appropriate value immediately, when the database configuration file is created, or when the database configuration file is reset.

Setting this parameter might help the optimizer model buffer pool usage more accurately. If you set this parameter manually, begin with a value of 2, regardless of the average number of applications that you run. After you assess the behavior of the optimizer and test the performance of your application at this setting, you can increase the value of the parameter in small increments. Continue to assess the behavior of the optimizer and test the performance of your application each time you increase the value of the parameter. Setting this parameter to a value that is too high might cause the optimizer to underestimate the amount of buffer pool space that is available to the query.

After you change the value of this parameter, consider rebinding applications by using the **REBIND PACKAGE** command.

backup_pending - Backup pending indicator

The **backup_pending** parameter indicates whether you need to do a full backup of the database before accessing it.

Configuration type
Database

Parameter type
Informational

This parameter is set to ON only if the database configuration is changed so that the database moves from being unrecoverable to recoverable. That is, initially both the **logarchmeth1** and **logarchmeth2** parameters were set to OFF, then either one or both of these parameters is set, and the update to the database configuration is accepted.

blk_log_dsk_ful - Block on log disk full

This parameter can be set to prevent disk full errors from being generated when the DB2 database system cannot create a new log file in the active log path.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
No [Yes; No]

Instead of generating a disk full error, the DB2 database system will attempt to create the log file every five minutes until it succeeds. After each attempt, the DB2 database system writes a message to the Administration Notification log. The only way that you can confirm that your application is hanging because of a log disk full condition is to monitor the Administration Notification log. Until the log file is successfully created, any user application that attempts to update table data will not be able to commit transactions. Read-only queries might not be directly affected; however, if a query needs to access data that is locked by an update request, or a data page that is fixed in the buffer pool by the updating application, read-only queries will also appear to hang.

Setting **blk_log_dsk_ful** to yes causes applications to hang when the DB2 database system encounters a log disk full error, thus allowing you to resolve the error and allowing the transaction to complete. You can resolve a full log disk situation by either increasing the file system capacity or by removing extraneous files on the same file system.

If **blk_log_dsk_ful** is set to no, then a transaction that receives a log disk full error will fail and be rolled back.

Set **blk_log_dsk_ful** to yes when using infinite logging; that is, when the database configuration parameter **logsecond** is set to -1. With infinite logging, the database may go offline in some situations if a transaction causes a log disk full error.

blocknonlogged - Block creation of tables that allow non-logged activity

This parameter specifies whether the database manager will allow tables to have the NOT LOGGED or NOT LOGGED INITIALLY attributes activated.

Configuration type
Database

Parameter type
Configurable Online

Configurable by member in a DB2 pureScale environment

Default [range]
`No [Yes, No ]`

By default, **blocknonlogged** is set to `NO`: non-logged operations are permitted and you gain the performance benefits associated with reduced logging. There are some potential drawbacks associated with this configuration, however, particularly in high availability disaster recovery (HADR) database environments. DB2 HADR database environments use database logs to replicate data from the primary database to the standby database. Non-logged operations are allowed on the primary database, but are not replicated to the standby database. If you perform any non-logged operations on the primary database, the standby database must be reinitialized. For example, you can use online split mirrors or suspended I/O support to resynchronize the standby database after non-logged operations.

Usage notes

- If **blocknonlogged** is set to `YES`, then CREATE TABLE and ALTER TABLE statements will fail if one of the following conditions is true:
  - The NOT LOGGED INITIALLY parameter is specified.
  - The NOT LOGGED parameter is specified for a LOB column.
  - A CLOB, DBCLOB, or BLOB column is defined as not logged.
- If **blocknonlogged** is set to `YES`, then the LOAD command fails if the following situations exist:
  - You specify the NONRECOVERABLE option.
  - You specify the COPY NO option.

cf_catchup_trgt - Target for catch up time of secondary cluster caching facility configuration parameter

This configuration parameter determines the target time in minutes for completing the catch up to bring a newly added or newly restarted cluster caching facility into peer state with an existing primary cluster caching facility .

Configuration type
Database

Parameter type
Configurable online

Default [range]
`15 [ 1 - 600 ]`

Unit of measure
Minutes since the secondary cluster caching facility was started

This parameter sets a target time for members to bring a newly added secondary cluster caching facility into peer state with the primary cluster caching facility. Once it is in peer state, the secondary cluster caching facility is able to take over the role of the primary cluster caching facility if the primary cluster caching facility fails or is shut down for maintenance. A lower setting for **cf_catchup_trgt** causes members to aggressively perform catch up activity, thus minimizing the window during which a DB2 pureScale instance is running without a secondary cluster caching facility that is ready to take over the role of the primary. A lower setting has a larger impact on database transactions and the overall performance of the system since more system resources (for example, I/O bandwidth) are used for catch up activity.

Note: Although the default setting for **cf_catchup_trgt** balances system performance with high availability, the following rule of thumb can be used to tune this parameter:

- If availability is more important than performance, use a lower setting for **cf_catchup_trgt**.
- If performance is more important than availability, use a higher setting for **cf_catchup_trgt**.

cf_db_mem_sz - Database memory configuration parameter

This parameter controls the total memory limit for the cluster caching facility, also designated as the CF, for this database.

Configuration type
Database

Applies to
Applies to a DB2 pureScale instance only.
- Database server with local and remote clients

Parameter type
Configurable online

Default [range]
`AUTOMATIC [32768 - 4 294 967 295]`

Unit of measure
Pages (4 KB)

The sum of the cluster caching facility structure memory limits for the **cf_gbp_sz**, **cf_lock_sz**, and **cf_sca_sz** parameters must be less than the CF structure memory limit for the **cf_db_mem_sz** parameter. Automatic memory tuning between CF resources such as the group buffer pool (GBP), shared communication area (SCA), and lock structures from within the same database is bound by the **cf_db_mem_sz** parameter.

Self-tuning of CF memory between databases is not supported.

Usage notes

- Set the value of the **numdb** configuration parameter higher than or equal to the total number of databases on the instance. Additionally, set the value of **DB2_DATABASE_CF_MEMORY** to a value that allows for every database on this instance, whether normally inactive or active, to be active at the same time.
- When manually setting the value of **cf_db_mem_sz**, the value must be less than the total CF server memory. The total CF server memory is controlled by the **cf_mem_sz** database manager configuration parameter.

- If the previous value of the **cf_db_mem_sz** parameter was set to AUTOMATIC, then the current value can be determined by using the **SHOW DETAIL** option of the **GET DATABASE CONFIGURATION** command.
- Use of the **cf_db_mem_sz** configuration parameter must be coordinated with the **DB2_DATABASE_CF_MEMORY** registry variable and the **numdb** configuration parameter.

Restrictions

- There is an additional 3840 4k pages taken from the **cf_mem_sz** parameter to be used internally by the CF. Additionally, there is an additional 256 4k pages reserved from **cf_mem_sz** for every active database in the instance. These additional pages only needs to be taken into consideration when setting **cf_mem_sz** manually.
- If altering the fixed value for this parameter to a new value that is *lower* than the current value, the new value must be greater than the sum of the memory sizes for the**cf_gbp_sz**, **cf_lock_sz**, and **cf_sca_sz** structure parameters or the operation will fail. To avoid this problem, lower the individual structure memory sizes before attempting to lower this parameter.
- If altering the fixed value for this parameter to a new value that is *higher* than the current value, there is an upper limit imposed, in addition to the total CF server memory defined by the **cf_mem_sz** parameter. Typically, this upper limit is defined by a memory buffer of between 3600 and 5000 pages that is required between the **cf_db_mem_sz** and **cf_mem_sz** parameter values. The value of the **cf_db_mem_sz** parameter should not exceed the difference of **cf_mem_sz** and this memory buffer.

cf_gbp_sz - Group buffer pool configuration parameter

This parameter determines the memory size used by the cluster caching facility, also known as CF, for group buffer pool (GBP) usage for this database.

Configuration type
Database

Applies to
Applies to a DB2 pureScale instance only.
- Database server with local and remote clients
- Client
- Database server with local clients

Parameter type
Configurable online

Default [range]
AUTOMATIC [AUTOMATIC, 4096 - 4294967296]

Unit of measure
Pages (4 KB)

When allocated
When the database is first connected or activated on any DB2 member

When freed
When the database is deactivated on the last DB2 member

There are two main types of resources in the GBP, directory entries and data elements. A directory entry stores metadata information pertaining to a page. A data element stores the actual page data.

DB2 members continue to cache pages in their own local buffer pools. The GBP is used only by the local buffer managers to maintain intersystem buffer coherency and cannot be directly referenced by DB2 EDUs running on any DB2 member.

cf_lock_sz - CF Lock manager configuration parameter

This parameter determines the memory size used by the CF for locking usage for this database.

Configuration type
Database

Applies to
Applies to a DB2 pureScale instance only.

- Database server with local and remote clients

Parameter type
Configurable online

Default [range]
AUTOMATIC [AUTOMATIC, 4096 - 1073741824]

Unit of measure
Pages (4 KB)

When allocated
When the database is first activated on any member.

Note: Memory is allocated on the CF and not on any of the members. If there is more than one CF then each CF allocates the same amount of memory.

When freed
When the database is deactivated on all members.

This is a global configuration parameter, meaning that the same value is used for every member in the DB2 pureScale instance.

Note that the memory on the CF is only allocated once; when the database is first activated on any of the members. You can monitor the consumption of the CF lock memory using the **current_cf_lock_size** monitor element.

cf_sca_sz - Shared communication area configuration parameter

This Shared Communication Area (SCA) configuration parameter determines the memory size used by the SCA in the cluster caching facility (CF). The SCA is a per database entity and contains database wide control block information for tables, indexes, table spaces, and catalogs.

Configuration type
Database

Applies to
DB2 pureScale environment

Parameter type
Configurable online

Default [range]
AUTOMATIC [AUTOMATIC, 2048 - 1073741824]

Unit of measure
Pages (4 KB)

When allocated
When the database is first activated on any DB2 member

When freed
When the database is dropped or when the CF server is stopped

When you set the **cf_sca_sz** parameter to AUTOMATIC (default), the memory value is computed by the DB2 database manager during the first database activation on any member to a value that is sufficient for basic database operations. If you want to configure the exact memory size for the SCA, you can set the **cf_sca_sz** parameter to a fixed numeric value.

You can obtain the current value for the SCA size by running the **GET DB CFG SHOW DETAIL** command. If the **cf_sca_sz** parameter is set to AUTOMATIC, the SHOW DETAIL clause displays the computed value as well as the allocated value for the SCA.

If the SCA memory is fully consumed by database operations, an out of memory error message is returned. In this case, you can increase the size of the SCA by setting a higher value for the **cf_sca_sz** parameter.

See "Configuring cluster caching facility memory for a database" for more detail.

catalogcache_sz - Catalog cache size

This parameter specifies the maximum space in pages that the catalog cache can use from the database heap.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
-1 [**maxappls***5, 8 - 524 288]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
When the database is initialized

When freed
When the database is shut down

This parameter is allocated out of the database shared memory, and is used to cache system catalog information. In a partitioned database system, there is one catalog cache for each database partition.

Caching catalog information at individual database partitions allows the database manager to reduce its processing time by eliminating the need to access the system catalogs (or the catalog node in a partitioned database environment) to obtain information that has previously been retrieved. The use of the catalog cache can help improve the overall performance of:

- Binding packages and compiling SQL and XQuery statements
- Operations that involve checking database-level privileges, routine privileges, global variable privileges and role authorizations
- Applications that are connected to non-catalog nodes in a partitioned database environment

By taking the default (-1) in a server or partitioned database environment, the value used to calculate the page allocation is five times the value specified for the **maxappls** configuration parameter. The exception to this occurs if five times **maxappls** is less than 8. In this situation, the default value of -1 will set **catalogcache_sz** to 8.

Recommendation: Start with the default value and tune it by using the database system monitor. When tuning this parameter, you should consider whether the extra memory being reserved for the catalog cache might be more effective if it was allocated for another purpose, such as the buffer pool or package cache.

Tuning this parameter is particularly important if a workload involves many SQL or XQuery compilations for a brief period of time, with few or no compilations thereafter. If the cache is too large, memory might be wasted holding copies of information that will no longer be used.

In an partitioned database environment, consider if the **catalogcache_sz** at the catalog node needs to be set larger since catalog information that is required at non-catalog nodes will always first be cached at the catalog node.

The **cat_cache_lookups** (catalog cache lookups), **cat_cache_inserts** (catalog cache inserts), **cat_cache_overflows** (catalog cache overflows), and **cat_cache_size_top** (catalog cache high water mark) monitor elements can help you determine whether you should adjust this configuration parameter.

Note: The catalog cache exists on all nodes in a partitioned database environment. Since there is a local database configuration file for each node, each node's **catalogcache_sz** value defines the size of the local catalog cache. In order to provide efficient caching and avoid overflow scenarios, you need to explicitly set the **catalogcache_sz** value at each node and consider the feasibility of possibly setting the **catalogcache_sz** on non-catalog nodes to be smaller than that of the catalog node; keep in mind that information that is required to be cached at non-catalog nodes will be retrieved from the catalog node's cache. Hence, a catalog cache at a non-catalog node is like a subset of the information in the catalog cache at the catalog node.

In general, more cache space is required if a unit of work contains several dynamic SQL or XQuery statements or if you are binding packages that contain a large number of static SQL or XQuery statements.

chngpgs_thresh - Changed pages threshold

This parameter specifies the level (percentage) of changed pages at which the asynchronous page cleaners will be started, if they are not currently active.

Configuration type
Database

Parameter type

- Configurable
- Configurable by member in a DB2 pureScale environment

Default [range]
60 [5 - 99]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Percentage

Asynchronous page cleaners will write changed pages from the buffer pool (or the buffer pools) to disk before the space in the buffer pool is required by a database agent. As a result, database agents should not have to wait for changed pages to be written out so that they might use the space in the buffer pool. This improves overall performance of the database applications.

When the page cleaners are started, they will build a list of the pages to write to disk. Once they have completed writing those pages to disk, they will become inactive again and wait for the next trigger to start.

When the **DB2_USE_ALTERNATE_PAGE_CLEANING** registry variable is set (that is, the alternate method of page cleaning is used), the **chngpgs_thresh** parameter has no effect, and the database manager automatically determines how many dirty pages to maintain in the buffer pool.

Recommendation: For databases with a heavy update transaction workload, you can generally ensure that there are enough clean pages in the buffer pool by setting the parameter value to be equal-to or less-than the default value. A percentage larger than the default can help performance if your database has a small number of very large tables.

codepage - Code page for the database

This parameter shows the code page that was used to create the database. The **codepage** parameter is derived based on the **codeset** parameter.

Configuration type
Database

Parameter type
Informational

codeset - Codeset for the database

This parameter shows the codeset that was used to create the database. Codeset is used by the database manager to determine **codepage** parameter values.

Configuration type
Database

Parameter type
Informational

collate_info - Collating information

This parameter determines the database's collating sequence. For a language-aware collation or locale-sensitive UCA collation, the first 256 bytes contain the string representation of the collation name (for example, "SYSTEM_819_US").

This parameter can only be displayed using the db2CfgGet API. It **cannot** be displayed through the command line processor.

Configuration type
Database

Parameter type
Informational

This parameter provides 260 bytes of database collating information. The first 256 bytes specify the database collating sequence, where byte “n” contains the sort weight of the code point whose underlying decimal representation is “n” in the code page of the database.

The last 4 bytes contain internal information about the type of the collating sequence. The last four bytes of the parameter is an integer. The integer is sensitive to the endian order of the platform. The possible values are:

- **0** - The sequence contains non-unique weights
- **1** - The sequence contains all unique weights
- **2** - The sequence is the identity sequence, for which strings are compared byte for byte.
- **3** - The sequence is NLSCHAR, used for sorting characters in a TIS620-1 (code page 874) Thai database.
- **4** - The sequence is IDENTITY_16BIT, which implements the “CESU-8 Compatibility Encoding Scheme for UTF-16: 8-bit” algorithm as specified in the Unicode Technical Report #26 available at the Unicode Technical Consortium website at http://www.unicode.org
- **X'8001'** - The sequence is UCA400_NO, which implements the Unicode Collation Algorithm (UCA) based on the Unicode Standard version 4.0.0, with normalization implicitly set to ON.
- **X'8002'** - The sequence is UCA400_LTH, which implements the Unicode Collation Algorithm (UCA) based on the Unicode Standard version 4.0.0, and sorts all Thai characters as per the Royal Thai Dictonary order.
- **X'8003'** - The sequence is UCA400_LSK, which implements the Unicode Collation Algorithm (UCA) based on the Unicode Standard version 4.0.0, and sorts all Slovakian characters properly.

Note:

- For a language-aware collation or locale-sensitive UCA collation, the first 256 bytes contain the string representation of the collation name.
-

  Important: Collations based on the Unicode Collation Algorithm of the Unicode Standard version 4.0.0 have been deprecated in Version 10.1 and might be removed in a future release. For more information, see “Collations based on the Unicode Collation Algorithm of the Unicode Standard version 4.0.0 have been deprecated” in *What’s New for DB2 Version 10.1*.

If you use this internal type information, you need to consider byte reversal when retrieving information for a database on a different platform.

You can specify the collating sequence at database creation time.

connect_proc - Connect procedure name database configuration parameter

This database configuration parameter allows you to input or update a two-part connect procedure name that will be executed every time an application connects to the database.

Configuration type
Database

Parameter type
Configurable Online

Configurable by member in a DB2 pureScale environment

Default
`NULL`

The following connect procedure conventions must be followed, otherwise an error is returned.

- The non-zero length string must specify a two-part procedure name (that is,[schema name].[procedure name])
- The connect procedure name (both schema and procedure name) can only contain the following characters:
  - A-Z
  - a-z
  - _ (underscore)
  - 0-9
- In addition, the schema and procedure name need to follow the rules of an ordinary identifier.

Once the **connect_proc** parameter is configured to a non-zero length value, the server will implicitly execute the procedure specified on every new connection.

Usage Notes

- A connection to the database is required when updating this parameter. However, unsetting the parameter does not require a connection if the database is deactivated.
- The **connect_proc** parameter can only be set using the **IMMEDIATE** option of the **UPDATE DATABASE CONFIGURATION** command. The **DEFERRED** option cannot be used when setting the **connect_proc** parameter.
- Only a procedure with exactly zero parameters can be used as a connect procedure. No other procedure sharing the same two-part name can exist in the database as long as the **connect_proc** parameter is set.
- The connect procedure must exist in the database before updating the **connect_proc** parameter. The **UPDATE DATABASE CONFIGURATION** command will fail with an error if the connect procedure with zero parameters does not exist in the database or if there is more than one procedure with the same name.
- Use the same connect procedure on all partitions in a data-partitioned environment.

country/region - Database territory code

This parameter shows the *territory* code used to create the database.

Configuration type
Database

Parameter type
Informational

database_consistent - Database is consistent

This parameter indicates whether the database is in a consistent state.

Configuration type
Database

Parameter type
Informational

`YES` indicates that all transactions have been committed or rolled back so that the data is consistent. If the system "crashes" while the database is consistent, you do not need to take any special action to make the database usable.

`NO` indicates that a transaction is pending or some other task is pending on the database and the data is not consistent at this point. If the system "crashes" while the database is not consistent, you will need to restart the database using the **RESTART DATABASE** command to make the database usable.

database_level - Database release level

This parameter indicates the release level of the database manager which can use the database.

Configuration type
Database

Parameter type
Informational

In the case of an incomplete or failed database upgrade, this parameter will reflect the release level of the database before the upgrade and might differ from the **release** parameter (the release level of the database configuration file). Otherwise the value of **database_level** will be identical to value of the **release** parameter.

database_memory - Database shared memory size

This parameter specifies the amount of memory that is reserved for the database shared memory region. If this amount is less than the amount calculated from the individual memory parameters (for example, locklist, utility heap, bufferpools, and so on), the larger amount will be used.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

  Note: If this parameter is set to different values in a DB2 pureScale environment, then **self_tuning_mem** should not be enabled.

Default [range]
Automatic [Computed, 0 - 4 294 967 295]

Unit of measure
Pages (4 KB)

When allocated
When the database is activated

When freed
When the database is deactivated

Setting this parameter to AUTOMATIC enables self-tuning. 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. For example, if the current database requirements are high, and there is sufficient free memory on the system, more memory will be consumed by database shared memory. Once the database memory requirements drop, or the amount of free memory on the system drops too low, some database shared memory is released.

The memory tuner will always leave a minimum amount of memory free based on the calculated benefit to providing additional memory to the instance. If there is a great benefit to providing an instance with more memory, then the memory tuner will maintain a lower amount of free memory. If the benefit is lower, then more free memory will be maintained. This allows databases to cooperate in the distribution of system memory.

Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self-tuning to be active.

Automatic tuning of this configuration parameter will only occur when self-tuning memory is enabled for the database (the **self_tuning_mem** configuration parameter is set to ON).

To simplify the management of this parameter, the COMPUTED setting instructs the database manager to calculate the amount of memory needed, and to allocate it at database activation time. The database manager will also allocate some additional memory to satisfy peak memory requirements for any heap in the database shared memory region whenever a heap exceeds its configured size. Other operations, such as dynamic configuration updates, also have access to this additional memory. The **db2pd** command, with the **-memsets** option, can be used to monitor the amount of unused memory left in the database shared memory region.

Recommendation: This value will usually remain at AUTOMATIC. For environments that do not support the AUTOMATIC setting, this should be set to COMPUTED. For example, the additional memory can be used for creating new buffer pools, or for increasing the size of existing buffer pools.

Note:

- In Version 9.7, when you set the **database_memory** configuration parameter to AUTOMATIC, the initial database shared memory allocation is the configured size of all heaps and buffer pools defined for the database, and the memory increases as needed. If **database_memory** is set to a specific value, then that requested amount of memory is allocated initially, during database activation. If the initial amount of memory cannot be allocated from the operating system, or exceeds the **instance_memory** limit, database activation fails with an SQL1084C error (Shared memory segments cannot be allocated).
- If you set **database_memory** to AUTOMATIC in DB2 Version 9.7 on Solaris Operating System, the database manager uses pageable memory for the database shared memory. In Solaris operating systems on UltraSPARC, the database manager attempts to use 64 KB memory pages if they are available. If 64 KB memory pages are not available, the database manager will use 8 KB memory pages. In Solaris operating systems on Sun x64 systems, the database manager will use 4 KB memory pages. The use of smaller memory pages might result in some performance degradation. There is also a greater requirement for swap space (equal to the size of database shared memory) due to the use of pageable shared memory.
- If you set **database_memory** to COMPUTED or a numeric value in DB2 Version 9.7 on Solaris, the database manager uses intimate shared memory (ISM) and large pages for the database shared memory.

Controlling DB2 Memory consumption:

When **instance_memory** is set to AUTOMATIC, a fixed upper bound on total memory consumption for the instance is set at instance startup (**db2start**). Actual memory consumption by the database manager varies depending on the workload. When self-tuning memory manager is enabled to perform **database_memory** tuning (by default for new databases), during runtime, self-tuning memory manager dynamically updates the size of performance-critical heaps within the database shared memory set according to the free physical memory on the system, while ensuring that there is sufficient free **instance_memory** available for functional memory requirements. For more information, see the **instance_memory** configuration parameter.

Limitation on some Linux[1] kernels:

Due to operating system limitations on some Linux kernels, self-tuning memory manager currently does not allow setting **database_memory** to AUTOMATIC. However, this setting is now allowed on these kernels only when **instance_memory** is set to a specific value, and not AUTOMATIC. If **database_memory** is set to AUTOMATIC, and **instance_memory** is later set back to AUTOMATIC, the **database_memory** configuration parameter is automatically updated to COMPUTED during the next database activation. If some databases are already active, self-tuning memory manager stops tuning the overall **database_memory** sizes.

[1] On Linux, this parameter supports the AUTOMATIC setting on RHEL5 and on SUSE 10 SP1 and newer, regardless of the setting of the **instance_memory** parameter. All other validated Linux distributions will return to COMPUTED if the kernel does not support this feature.

dbheap - Database heap

This parameter determines the maximum memory used by the database heap.

With Version 9.5, this database configuration parameter has a default value of AUTOMATIC, meaning that the database heap can increase as needed until either the **database_memory** limit is reached, or the **instance_memory** limit is reached.

Configuration type
Database

Parameter type
- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
Automatic [32 - 524 288]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
When the database is activated

When freed
When the database is deactivated

There is one database heap per database, and the database manager uses it on behalf of all applications connected to the database. It contains control block information for tables, indexes, table spaces, and buffer pools. It also contains space for the log buffer (**logbufsz**) and temporary memory used by utilities. Therefore, the size of the heap will be dependent on a large number of variables. The control block information is kept in the heap until all applications disconnect from the database.

The minimum amount the database manager needs to get started is allocated at the first connection. The data area is expanded as needed until either the configured upper limit is reached, or, if set to AUTOMATIC, until all **database_memory** or **instance_memory**, or memory for both, is exhausted.

The following formula can be used as a rough guideline when deciding on a value to assign to the **dbheap** configuration parameter:

```
10K per tablespace + 4K per table + (1K + 4*extents used),
  per range clustered table (RCT)
```

The **dbheap** value that you configure represents only a portion of the database heap that is allocated. The database heap is the main memory area used to satisfy global database memory requirements. It is sized to include basic allocations needed for the startup of a database in addition to the **dbheap** value. Tools which report memory usage such as Memory Tracker, Snapshot Monitor, and **db2pd** report the statistics of this larger database heap. There is no separate tracking of the allocations that are represented by the **dbheap** configuration parameter. Therefore, it is normal for the statistics on database heap memory usage reported from these tools to exceed the configured value for the **dbheap** parameter.

You can use the database system monitor to track the highest amount of memory that was used for the database heap, using the **db_heap_top** (maximum database heap allocated) element.

Note:

- Workload Management (WLM) work class sets and work action sets are stored in the database heap. However, a very small part of the memory is consumed for this.
- Trusted contexts, Workload Management, and Audit policy information is cached in memory for fast processing. This memory is allocated from the database heap. Therefore, user-defined trusted contexts, workload management, and audit policy objects would impose more memory requirements on the database heap. In this case, it is suggested that you set your database heap configuration to `AUTOMATIC` so that the database manager automatically manages the database heap size.

db_mem_thresh - Database memory threshold

This parameter represents the maximum percentage of committed, but currently unused, database shared memory that the database manager will allow before starting to release committed pages of memory back to the operating system.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
`10 [0-100 ]`

Unit of measure
Percentage

This database configuration parameter relates to how the database manager handles excess unused database shared memory. Typically, as pages of memory are touched by a process, they are committed, meaning that a page of memory has been allocated by the operating system and occupies space either in physical memory or in a page file on disk. Depending on the database workload, there might be peak database shared memory requirements at a certain times of day. Once the operating system has enough committed memory to meet those peak requirements, that memory remains committed, even after peak memory requirements have subsided.

Acceptable values are whole numbers in the range of `0` (immediately release any unused database shared memory) to `100` (never release any unused database shared memory). The default is `10` (release unused memory only when more than 10% of database shared memory is currently unused), which should be suitable for most workloads.

This configuration parameter can be updated dynamically. Care should be taken when updating this parameter, as setting the value too low could cause excessive memory thrashing on the box (memory pages constantly being committed and

then released), and setting the value too high might prevent the database manager from returning any database shared memory back to the operating system for other processes to use.

This configuration parameter will be ignored (meaning that unused database shared memory pages will remain committed) if the database shared memory region is pinned through the **DB2_PINNED_BP** registry variable, configured for large pages through the **DB2_LARGE_PAGE_MEM** registry variable, or if releasing of memory is explicitly disabled through the **DB2MEMDISCLAIM** registry variable.

Some versions of Linux do not support releasing subranges of a shared memory segment back to the operating system. On such platforms, this parameter will be ignored.

date_compat - Date compatibility database configuration parameter

This parameter indicates whether the DATE compatibility semantics associated with the TIMESTAMP(0) data type are applied to the connected database.

Configuration type
Database

Parameter type
Informational

The value is determined at database creation time, and is based on the setting of the **DB2_COMPATIBILITY_VECTOR** registry variable for DATE data type. The value cannot be changed.

dec_to_char_fmt - Decimal to character function configuration parameter

This parameter is used to control the result of the CHAR scalar function and the CAST specification for converting decimal to character values.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

See Consequences of changing dec_to_char_fmt.

Default [range]
NEW [NEW, V95]

The setting of the parameter determines whether leading zeros and a trailing decimal characters are included in the result of the CHAR function. If you set the parameter to `NEW`, leading zeros and a trailing decimal characters are not included; if you set the parameter to `V95`, leading zeros and a trailing decimal characters are included.

Leading zeros and a trailing decimal characters are also included in the result of the CHAR_OLD scalar function, which has the same syntax as the CHAR function.

When upgrading, for databases created before Version 9.7 and then upgraded to Version 9.7 or higher, the parameter **dec_to_char_fmt** is set to *V95* by default.

Effects of changing the value of dec_to_char_fmt

- Materialized query tables (MQTs) that you created before Version 9.7 might contain results that differ from those MQTs that you created by using the NEW setting. To ensure that previously created MQTs contain only data that adheres to the new format, refresh these MQTs by using the REFRESH TABLE statement.
- The results of a trigger may be affected by the changed format. Setting the value of the parameter to NEW to change the format has no effect on data that has already been written.
- Constraints that allowed data to be inserted into a table might, if reevaluated, reject that same data. Similarly, constraints that did not allow data to be inserted into a table might, if reevaluated, accept that same data. Use the SET INTEGRITY statement to check for and correct data in a table that might no longer satisfy a constraint.
- After changing the value of **dec_to_char_fmt**, recompile all static SQL packages that depend on the value of a generated column whose results are effected by the change in the **dec_to_char_fmt** value. To find out which static SQL packages are effected, you must compile, rebind all the packages using the **db2rbind** command.

decflt_rounding - Decimal floating point rounding configuration parameter

This parameter allows you to specify the rounding mode for decimal floating point (DECFLOAT). The rounding mode affects decimal floating-point operations in the server, and in **LOAD**.

Configuration type
Database

Parameter type
Configurable

See "Consequences of changing decflt_rounding" on page 769.

Default [range]
ROUND_HALF_EVEN [ROUND_CEILING, ROUND_FLOOR, ROUND_HALF_UP, ROUND_DOWN]

DB2 database systems support five IEEE-compliant decimal floating point rounding modes. The rounding mode specifies how to round the result of a calculation when the result exceeds the precision. The definitions for all the rounding modes are as follows:

ROUND_CEILING
Round towards +infinity. If all of the discarded digits are zero or if the sign is negative the result is unchanged. Otherwise, the result coefficient should be incremented by 1 (rounded up).

ROUND_FLOOR
Round towards -infinity. If all of the discarded digits are zero or if the sign is positive the result is unchanged. Otherwise, the sign is negative and the result coefficient should be incremented by 1.

ROUND_HALF_UP
Round to nearest; if equidistant, round up 1. If the discarded digits represent greater than or equal to half (0.5) of the value of a 1 in the next left position then the result coefficient should be incremented by 1 (rounded up). Otherwise, the discarded digits (0.5 or less) are ignored.

ROUND_HALF_EVEN
Round to nearest; if equidistant, round so that the final digit is even. If the discarded digits represent greater than half (0.5) the value of a one in the next left position, then the result coefficient should be increment by 1 (rounded up). If they represent less than half, then the result coefficient is not adjusted, that is, the discarded digits are ignored. Otherwise, if they represent exactly half, the result coefficient is unaltered if its rightmost digit is even, or incremented by 1 (rounded up) if its rightmost digit is odd, to make an even digit. This rounding mode is the default rounding mode as per IEEE decimal floating point specification and is the default rounding mode in DB2 database products.

ROUND_DOWN
Round towards 0 (truncation). The discarded digits are ignored.

Table 136 shows the result of rounding of 12.341, 12.345, 12.349, 12.355, and -12.345, each to 4 digits, under different rounding modes:

Table 136. Decimal floating point rounding modes

Rounding mode	12.341	12.345	12.349	12.355	-12.345
ROUND_DOWN	12.34	12.34	12.34	12.35	-12.34
ROUND_HALF_UP	12.34	12.35	12.35	12.36	-12.35
ROUND_HALF_EVEN	12.34	12.34	12.35	12.36	-12.34
ROUND_FLOOR	12.34	12.34	12.34	12.35	-12.35
ROUND_CEILING	12.35	12.35	12.35	12.36	-12.34

Consequences of changing decflt_rounding

- Previously constructed materialized query tables (MQTs) could contain results that differ from what would be produced with the new rounding mode. To correct this problem, refresh potentially impacted MQTs.
- The results of a trigger may be affected by the new rounding mode. Changing it has no effect on data that has already been written.
- Constraints that allowed data to be inserted into a table, if reevaluated, might reject that same data. Similarly constraints that did not allow data to be inserted into a table, if reevaluated, might accept that same data. Use the SET INTEGRITY statement to check for and correct such problems. The value of a generated column whose calculation is dependent on **decflt_rounding** could be different for two identical rows except for the generated column value, if one row was inserted before the change to **decflt_rounding** and the other was inserted after.
- The rounding mode is not compiled into sections. Therefore, static SQL does not need to be recompiled after changing **decflt_rounding**.

Note: The value of this configuration parameter is not changed dynamically but will become effective only after all applications disconnect from the database. If the database is activated, it must be deactivated.

dft_degree - Default degree

This parameter specifies the default value for the CURRENT DEGREE special register and the **DEGREE** bind option.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Connection

Default [range]
1 [-1(ANY), 1 - 32 767]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

The default value is 1.

A value of 1 means no intrapartition parallelism. A value of -1 (or ANY) means the optimizer determines the degree of intrapartition parallelism based on the number of processors and the type of query.

The degree of intrapartition parallelism for an SQL statement is specified at statement compilation time using the CURRENT DEGREE special register or the **DEGREE** bind option. The maximum runtime degree of intrapartition parallelism for an active application is specified using the **SET RUNTIME DEGREE** command. The Maximum Query Degree of Parallelism (**max_querydegree**) configuration parameter specifies the maximum query degree of intrapartition parallelism for all SQL queries.

The actual runtime degree used is the lowest of:

- **max_querydegree** configuration parameter
- application runtime degree
- SQL statement compilation degree

dft_extent_sz - Default extent size of table spaces

This parameter sets the default extent size of table spaces.

Configuration type
Database

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
32 [2 - 256]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages

When a table space is created, EXTENTSIZE n can be optionally specified, where n is the extent size. If you do not specify the extent size on the CREATE TABLESPACE statement, the database manager uses the value given by this parameter.

Recommendation: In many cases, you will want to explicitly specify the extent size when you create the table space. Before choosing a value for this parameter, you should understand how you would explicitly choose an extent size for the CREATE TABLESPACE statement.

In a DB2 pureScale environment, you should use an extent size of at least the default (32 pages). This minimum extent size reduces the amount of internal message traffic within the DB2 pureScale environment when extents are added for a table or index. Examples of cases where new extents are allocated frequently, and where a larger extent size is beneficial, include the load and import utilities, the CREATE INDEX statement, and bulk insert operations from applications.

dft_loadrec_ses - Default number of load recovery sessions

This parameter specifies the default number of sessions that will be used during the recovery of a table load.

Configuration type
Database

Parameter type

- Configurable online

Propagation class
Immediate

Default [range]
1 [1 - 30 000]

Unit of measure
Counter

The value of this parameter should be set to the number of I/O sessions that was specified with the `COPY YES` option in the original **`LOAD`** command. The retrieval of a load copy is an operation similar to restore. You can override this parameter through entries in the copy location file specified by the environment variable **`DB2LOADREC`**.

The default number of buffers used for load retrieval is two more than the value of this parameter. You can also override the number of buffers in the copy location file.

This parameter is applicable only if roll forward recovery is enabled.

dft_mttb_types - Default maintained table types for optimization

This parameter specifies the default value for the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register. The value of this register determines what types of refresh deferred materialized query tables will be used during query optimization.

Configuration type
Database

Parameter type
Configurable

Default [range]
SYSTEM [ALL, NONE, FEDERATED_TOOL, SYSTEM, USER, or a list of values]

You can specify a list of values separated by commas; for example, 'USER,FEDERATED_TOOL'. ALL or NONE cannot be listed with other values, and you cannot specify the same value more than once. For use with the **db2CfgSet** and **db2CfgGet** APIs, the acceptable parameter values are: 8 (ALL), 4 (NONE), 16 (FEDERATED_TOOL), 1 (SYSTEM) and 2 (USER). Multiple values can be specified together using bitwise OR; for example, 18 would be the equivalent of USER,FEDERATED_TOOL. As before, the values 4 and 8 cannot be used with other values.

dft_prefetch_sz - Default prefetch size

This parameter sets the default prefetch size of table spaces.

Configuration type
Database

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
`Automatic [0 - 32 767]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages

When a table space is created, PREFETCHSIZE can optionally be specified with a value of AUTOMATIC or *n*, where *n* represents the number of pages the database manager will read if prefetching is being performed. If you do not specify the prefetch size on invocation of the CREATE TABLESPACE statement, the database manager uses the current value of the **dft_prefetch_sz** parameter.

If a table space is created with the prefetch size set to `AUTOMATIC`, the DB2 database manager will automatically calculate and update the prefetch size of the table space.

This calculation is performed:

- When the database starts
- When a table space is first created with AUTOMATIC prefetch size
- When the number of containers for a table space changes through execution of an ALTER TABLESPACE statement
- When the prefetch size for a table space is updated to be AUTOMATIC through execution of an ALTER TABLESPACE statement

The `AUTOMATIC` state of the prefetch size can be turned on or off as soon as the prefetch size is updated manually through invocation of the ALTER TABLESPACE statement.

Recommendation: Using system monitoring tools, you can determine if your CPU is idle while the system is waiting for I/O. Increasing the value of this parameter can help if the table spaces being used do not have a prefetch size defined for them.

This parameter provides the default for the entire database, and it might not be suitable for all table spaces within the database. For example, a value of 32 might be suitable for a table space with an extent size of 32 pages, but not suitable for a table space with an extent size of 25 pages. Ideally, you should explicitly set the prefetch size for each table space.

To help minimize I/O for table spaces defined with the default extent size (**dft_extent_sz**), you should set this parameter as a factor or whole multiple of the value of the **dft_extent_sz** parameter. For example, if the **dft_extent_sz** parameter is 32, you could set **dft_prefetch_sz** to 16 (a fraction of 32) or to 64 (a whole multiple of 32). If the prefetch size is a multiple of the extent size, the database manager might perform I/O in parallel, if the following conditions are true:

- The extents being prefetched are on different physical devices
- Multiple I/O servers are configured (**num_ioservers**).

dft_queryopt - Default query optimization class

The query optimization class is used to direct the optimizer to use different degrees of optimization when compiling SQL and XQuery queries. This parameter provides additional flexibility by setting the default query optimization class used when neither the SET CURRENT QUERY OPTIMIZATION statement nor the **QUERYOPT** option on the **BIND** command are used.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Connection

Default [range]
5 [0 - 9]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Query Optimization Class (see the following list)

The query optimization classes currently defined are:

- 0 - minimal query optimization.
- 1 - roughly comparable to DB2 Version 1.
- 2 - slight optimization.
- 3 - moderate query optimization.
- 5 - significant query optimization with heuristics to limit the effort expended on selecting an access plan. This is the default.
- 7 - significant query optimization.

- 9 - maximal query optimization

dft_refresh_age - Default refresh age

This parameter represents the maximum time duration since a REFRESH TABLE statement has been processed on a specific REFRESH DEFERRED materialized query table. After this time limit is exceeded, the materialized query table is not used to satisfy queries until the materialized query table is refreshed.

Configuration type
Database

Parameter type
Configurable

Default [range]
0 [0, 99999999999999 (ANY)]

Unit of measure
Seconds

This parameter has the default value used for the REFRESH AGE if the CURRENT REFRESH AGE special register is not specified. This parameter specifies a time stamp duration value with a data type of DECIMAL(20,6). If the CURRENT REFRESH AGE has a value of 99999999999999 (ANY), and the QUERY OPTIMIZATION class has a value of two, five or more, REFRESH DEFERRED materialized query tables are considered to optimize the processing of a dynamic query.

dft_schemas_dcc - Default data capture on new schemas configuration parameter

This parameter allows the control of default setting for DATA CAPTURE CHANGES on newly created schemas for replication purposes.

Configuration type
Database

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
No [Yes; No]

By default, **dft_schemas_dcc** is set to NO. When set to YES, all newly created schemas by default will have the DATA CAPTURE CHANGES clause.

dft_sqlmathwarn - Continue upon arithmetic exceptions

This parameter sets the default value that determines the handling of arithmetic errors and retrieval conversion errors as errors or warnings during SQL statement execution.

Configuration type
Database

Parameter type
Configurable

Default [range]
No [No, Yes]

For static SQL statements, the value of this parameter is associated with the package at bind time. For dynamic SQL DML statements, the value of this parameter is used when the statement is prepared.

Attention: If you change the **dft_sqlmathwarn** value for a database, the behavior of check constraints, triggers, and views that include arithmetic expressions might change. This might, in turn, have an impact on the data integrity of the database. You should only change the setting of **dft_sqlmathwarn** for a database after carefully evaluating how the new arithmetic exception handling behavior might impact check constraints, triggers, and views. Once changed, subsequent changes require the same careful evaluation.

As an example, consider the following check constraint, which includes a division arithmetic operation:

```
A/B > 0
```

When **dft_sqlmathwarn** is No and an INSERT with B=0 is attempted, the division by zero is processed as an arithmetic error. The insert operation fails because the DB2 database manager cannot check the constraint. If **dft_sqlmathwarn** is changed to Yes, the division by zero is processed as an arithmetic warning with a NULL result. The NULL result causes the predicate to evaluate to UNKNOWN and the insert operation succeeds. If **dft_sqlmathwarn** is changed back to No, an attempt to insert the same row will fail, because the division by zero error prevents the DB2 database manager from evaluating the constraint. The row inserted with B=0 when **dft_sqlmathwarn** was Yes remains in the table and can be selected. Updates to the row that cause the constraint to be evaluated will fail, while updates to the row that do not require constraint re-evaluation will succeed.

Before changing **dft_sqlmathwarn** from No to Yes, you should consider rewriting the constraint to explicitly handle nulls from arithmetic expressions. For example:

```
( A/B > 0 ) AND ( CASE
                    WHEN A IS NULL THEN 1
                    WHEN B IS NULL THEN 1
                    WHEN A/B IS NULL THEN 0
                    ELSE 1
                    END
                  = 1 )
```

can be used if both A and B are nullable. And, if A or B is not-nullable, the corresponding IS NULL WHEN-clause can be removed.

Before changing **dft_sqlmathwarn** from Yes to No, you should first check for data that might become inconsistent by using, for example, predicates such as the following:

```
WHERE A IS NOT NULL AND B IS NOT NULL AND A/B IS NULL
```

When inconsistent rows are isolated, you should take appropriate action to correct the inconsistency before changing **dft_sqlmathwarn**. You can also manually re-check constraints with arithmetic expressions after the change. To do this, first place the affected tables in a check pending state (with the OFF clause of the SET CONSTRAINTS statement), then request that the tables be checked (with the

IMMEDIATE CHECKED clause of the SET CONSTRAINTS statement). Inconsistent data will be indicated by an arithmetic error, which prevents the constraint from being evaluated.

Recommendation: Use the default setting of `no`, unless you specifically require queries to be processed that include arithmetic exceptions. Then specify the value of `yes`. This situation can occur if you are processing SQL statements that, on other database managers, provide results regardless of the arithmetic exceptions that occur.

discover_db - Discover database

This parameter is used to prevent information about a database from being returned to a client when a discovery request is received at the server.

Configuration type
Database

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
`ENABLE [DISABLE, ENABLE]`

The default for this parameter is that discovery is enabled for this database.

By changing this parameter value to `Disable`, it is possible to hide databases with sensitive data from the discovery process. This can be done in addition to other database security controls on the database.

dlchktime - Time interval for checking deadlock

This parameter defines the frequency at which the database manager checks for deadlocks among all the applications connected to a database.

Configuration type
Database

Parameter type
Configurable online

Propagation class
Immediate

Default [range]
`10 000` (10 seconds) `[1 000 - 600 000]`

Unit of measure
Milliseconds

A deadlock occurs when two or more applications connected to the same database wait indefinitely for a resource. The waiting is never resolved because each application is holding a resource that the other needs to continue.

Note:

1. In a partitioned database environment, this parameter applies to the catalog node only.

2. In a partitioned database environment, a deadlock is not flagged until after the second iteration.

Recommendation: Increasing this parameter decreases the frequency of checking for deadlocks, thereby increasing the time that application programs must wait for the deadlock to be resolved.

Decreasing this parameter increases the frequency of checking for deadlocks, thereby decreasing the time that application programs must wait for the deadlock to be resolved but increasing the time that the database manager takes to check for deadlocks. If the deadlock interval is too small, it can decrease runtime performance, because the database manager is frequently performing deadlock detection. If this parameter is set lower to improve concurrency, you should ensure that **maxlocks** and **locklist** are set appropriately to avoid unnecessary lock escalation, which can result in more lock contention and as a result, more deadlock situations.

enable_xmlchar - Enable conversion to XML configuration parameter

This parameter determines whether XMLPARSE operations can be performed on non-BIT DATA CHAR (or CHAR-type) expressions in an SQL statement.

Configuration type
Database

Parameter type
Configurable

Default [range]
`Yes [Yes; No]`

When pureXML® features are used in a non-Unicode database, the XMLPARSE function can cause character substitutions to occur as SQL string data is converted from the client code page into the database code page, and then into Unicode for internal storage. Setting **enable_xmlchar** to `NO` blocks the usage of character data types during XML parsing, and any attempts to insert character types into a non-Unicode database will generate an error. The BLOB data type and FOR BIT DATA data types are still allowed when **enable_xmlchar** is set to `NO`, as code page conversion does not occur when these data types are used to pass XML data into a database.

By default, **enable_xmlchar** is set to `YES` so that parsing of character data types is allowed. In this case, you should ensure that any XML data to be inserted contains only code points that are part of the database code page, in order to avoid substitution characters being introduced during insertion of the XML data.

Note: The client needs to disconnect and reconnect to the agent for this change to be reflected.

failarchpath - Failover log archive path

This parameter specifies a path to which the DB2 database system will try to archive log files if the log files cannot be archived to either the primary or the secondary (if set) archive destinations because of a media problem affecting those destinations. This specified path must reference a disk.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Default [range]
Null [ ]

If there are log files in the path specified by the current value of **failarchpath**, any updates to **failarchpath** will not take effect immediately. Instead, the update will take effect when all applications disconnect.

groupheap_ratio - Percent of memory for application group heap

This parameter has been deprecated since Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the database manager in DB2 Version 9.5 or later releases. In Version 9.5, it has been replaced by the **appl_memory** configuration parameter.

Note: The following information applies only to pre-Version 9.5 data servers and clients.

This parameter specifies the percentage of memory in the application control shared memory set devoted to the application group shared heap.

Configuration type
Database

Parameter type
Configurable

Default [range]
70 [1 - 99]

Unit of measure
Percentage

This parameter does not have any effect on a non-partitioned database with concentrator OFF and intrapartition parallelism disabled.

Recommendation: Retain the default value of this parameter unless you are experiencing performance problems.

hadr_db_role - HADR database role

This parameter indicates the current role of a database, whether the database is online or offline.

Configuration type
Database

Applies to

- Database server with local and remote clients

- Database server with local clients

Parameter type
Informational

Valid values are: STANDARD, PRIMARY, or STANDBY.

Note: When a database is active, the HADR role of the database can also be determined using the **GET SNAPSHOT FOR DATABASE** command.

hadr_local_host - HADR local host name

This parameter specifies the local host for high availability disaster recovery (HADR) TCP communication.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients

Parameter type

- Configurable[6]

Default
Null

Either a host name or an IP address can be used. If a host name is specified and it maps to multiple IP addresses, an error is returned, and HADR will not start up. If the host name maps to multiple IP addresses (even if you specify the same host name on primary and standby), primary and standby can end up mapping this host name to different IP addresses, because some DNS servers return IP address lists in non-deterministic order.

A host name is in the form: myserver.ibm.com. An IP address is in the form: "12.34.56.78".

Usage Notes

- If your primary and standby start but do not connect to one another, and no error indication appears in the DB2 diagnostic log, it could be due to subtle issues in host name resolution. If you configured HADR using host names, try again using the IP addresses instead.
- If either of the primary or standby resides in a private network behind a NAT (Network Address Translation) device, you may need to set the registry variable **DB2_HADR_NO_IP_CHECK** to ON. Refer to HADR and Network Address Translation (NAT) support

hadr_local_svc - HADR local service name

This parameter specifies the TCP service name or port number for which the local high availability disaster recovery (HADR) process accepts connections.

Configuration type
Database

6. Changes to this parameter take effect on database activation. If the database is already online, you can have changes take effect by stopping and restarting HADR on the primary database.

Applies to

- Database server with local and remote clients
- Database server with local clients

Parameter type

- Configurable[7]

Default

Null

The value for **hadr_local_svc** on the Primary or Standby database systems cannot be the same as the value of **svcename** or **svcename** +1 on their corresponding hosts.

If you are using SSL, do not set **hadr_local_svc** on the Primary or Standby database system to the same value as you set for **ssl_svcename**.

hadr_peer_window - HADR peer window configuration parameter

When you set **hadr_peer_window** to a non-zero time value, then a HADR primary-standby database pair continues to behave as though still in peer state, for the configured amount of time, if the primary database loses connection with the standby database. This helps ensure data consistency.

Configuration type

Database

Parameter type

- Configurable[8]

Default [range]

0 [0 — 4 294 967 295]

Unit of measure

Seconds

Usage notes:

- The value needs to be the same on both primary and standby databases, unless you are using HADR multiple standby mode. With multiple standbys, the principal standby uses the primary's setting for **hadr_peer_window** and any setting for **hadr_peer_window** on the auxiliary standbys is ignored.
- A recommended minimum value is 120 seconds.
- When the **hadr_syncmode** value is set to ASYNC or SUPERASYNC, the **hadr_peer_window** value is ignored.
- To avoid impacting the availability of the primary database when the standby database is intentionally shut down, for example, for maintenance, the peer window is not invoked if the standby database is explicitly deactivated while the HADR pair is in peer state.
- The **TAKEOVER HADR** command with the **PEER WINDOW ONLY** option launches a takeover operation only if the HADR standby is presently inside the defined peer window.

7. Changes to this parameter take effect on database activation. If the database is already online, you can have changes take effect by stopping and restarting HADR on the primary database.

8. Changes to this parameter take effect on database activation. If the database is already online, you can have changes take effect by stopping and restarting HADR on the primary database.

- The takeover operation with the **hadr_peer_window** parameter may behave incorrectly if the primary database clock and the standby database clock are not synchronized to within 5 seconds of each other. That is, the operation may succeed when it should fail, or fail when it should succeed. You should use a time synchronization service (for example, NTP) to keep the clocks synchronized to the same source.
- On the standby database, the peer window end time is a time specified in the last heartbeat message that the standby received from the primary database, and is not directly related to when the standby detects loss of the connection.

hadr_remote_host - HADR remote host name

This parameter specifies the TCP/IP host name or IP address of the remote high availability disaster recovery (HADR) database server.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients

Parameter type

- Configurable[9]

Default
Null

Similar to **hadr_local_host**, this parameter must map to only one IP address.

hadr_remote_inst - HADR instance name of the remote server

This parameter specifies the instance name of the remote server. High availability disaster recovery (HADR) also checks whether a remote database requesting a connection belongs to the declared remote instance.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients

Parameter type

- Configurable[10]

Default
Null

hadr_remote_svc - HADR remote service name

This parameter specifies the TCP service name or port number that will be used by the remote high availability disaster recovery (HADR) database server.

9. Changes to this parameter take effect on database activation. If the database is already online, you can have changes take effect by stopping and restarting HADR on the primary database.

10. Changes to this parameter take effect on database activation. If the database is already online, you can have changes take effect by stopping and restarting HADR on the primary database.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients

Parameter type

- Configurable[11]

Default
Null

hadr_replay_delay - HADR replay delay configuration parameter

This parameter specifies the number of seconds that must pass from the time that a transaction is committed on the primary database to the time that the transaction is committed on the standby database.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients

Parameter type
Configurable

Default [Range]
0 [0 to 2147483647]

Unit of measure
Seconds

The **hadr_replay_delay** configuration parameter enables *delayed replay* on the HADR standby database, meaning that the standby intentionally lags the HADR primary database as the logs from the primary are replayed. The standby database must be in SUPERASYNC mode before **hadr_replay_delay** can be set to a nonzero value. You cannot set the parameter on the primary database. You must disable delayed replay on the standby before it can take over as the primary.

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, replay position can be far behind log receive position on standby. Without spooling, log receive can only go beyond replay by the amount of the receive buffer. With spooling, the standby can receive much more logs beyond 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 will not be blocked by the delayed replay.

With delayed replay, if there is an errant transaction on the primary and it is noticed before it is replayed on the standby, you can roll forward the database to a time that is right before when the errant transaction is committed, then stop roll forward to retrieve data lost on the primary.

11. Changes to this parameter take effect on database activation. If the database is already online, you can have changes take effect by stopping and restarting HADR on the primary database.

Note: If you have delayed replay enabled on a standby, that standby cannot take over as the new primary until you have disabled delayed replay (set the **hadr_replay_delay** parameter to 0 on that standby).

hadr_spool_limit - HADR log spool limit configuration parameter

This parameter determines the maximum amount of log data that is allowed to be spooled to disk on HADR standby.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients

Parameter type
Configurable[12]

Default [range]
0 [-1 - 2 147 483 647]

Unit of measure
Pages (4 KB)

The **hadr_spool_limit** configuration parameter enables log spooling on the HADR standby database. Log spooling allows transactions on the HADR primary to make progress without having to wait for the log replay on HADR standby. Log data that is sent by the primary is then written, or *spooled*, to disk on the standby if it falls behind in log replay. The standby can later on read the log 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.

When making use of log spooling, ensure that adequate disk space is provided to the active log path of the standby database. There must be enough disk space to hold the active logs, which is determined by the **logprimary**, **logsecond**, **logfilsiz**, and **hadr_spool_limit** configuration parameters.

The default value of 0 means no spooling. This means that the standby can be behind the primary up to the size up the log receive buffer. When the buffer is full, it is possible that new transactions on the primary will be blocked because the primary cannot send any more log data to the standby system.

A value of -1 means unlimited spooling (as much as supported by the disk space available). If you are using a high value for **hadr_spool_limit**, you should consider that if there is a large gap between the log position of the primary and log replay on the standby, which might lead to a longer takeover time because the standby cannot assume the role of the new standby until the replay of the spooled logs finishes.

Note that making use of log spooling does not compromise the HADR protection provided by the HADR feature. Data from the primary is still replicated in log

12. Changes to this parameter take effect on database activation. If the database is already online, you can have changes take effect by stopping and restarting HADR on the primary database.

form to the standby using the specified sync mode; it just takes time to apply (through log replay) the data to the table spaces.

hadr_syncmode - HADR synchronization mode for log write in peer state

This parameter specifies the synchronization mode, which determines how primary log writes are synchronized with the standby when the systems are in peer state.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients

Parameter type

- Configurable[13]

Default [range]
NEARSYNC [ASYNC, SUPERASYNC, SYNC]

Valid values for this parameter are:

SYNC This mode provides the greatest protection against transaction loss, but at a higher cost of transaction response time.

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.

NEARSYNC
This mode provides somewhat less protection against transaction loss, in exchange for a shorter transaction response time than that of SYNC mode.

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 standby site has not transferred to nonvolatile storage all of the log data that it has received.

ASYNC
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 this 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.

13. Changes to this parameter take effect on database activation. If the database is already online, you can have changes take effect by stopping and restarting HADR on the primary database.

SUPERASYNC

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 only when the log records have been written to the log files on the primary database. 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.

Figure 53 shows when the logs for transactions are considered successful based on the synchronization mode chosen:

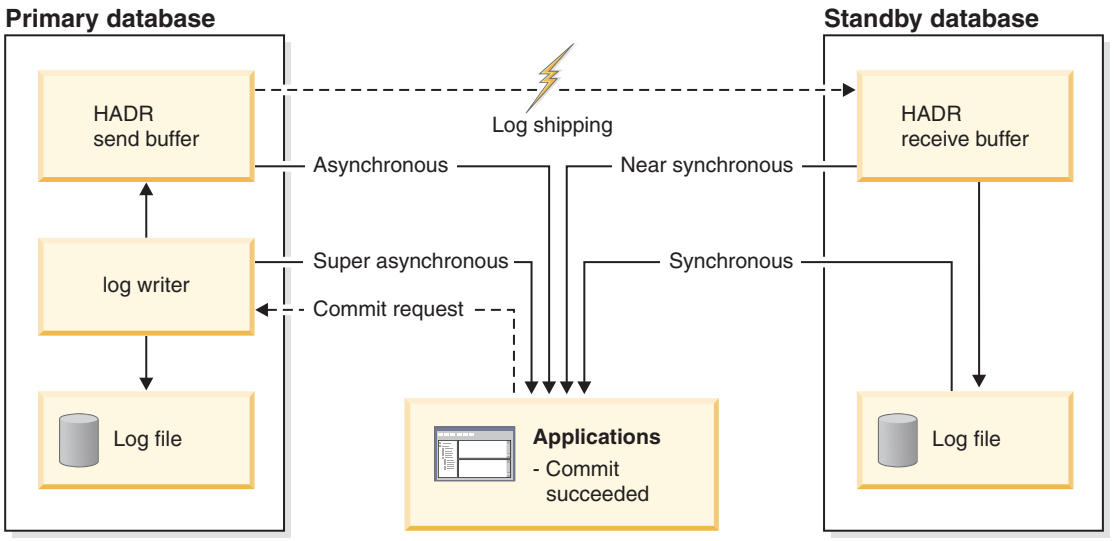


Figure 53. Synchronization modes for high availability and disaster recovery (HADR)

hadr_target_list - HADR target list database configuration parameter

This parameter, which enables HADR to run in multiple standby mode, specifies a list of up to three target *host*:*port* pairs that act as HADR standby databases.

Configuration type

Database

Applies to

- Database server with local and remote clients
- Database server with local clients

Parameter type

Configurable online

Default

NULL

For HADR to run in multiple standby mode, you must set the **hadr_target_list** configuration parameter on all HADR databases. You cannot switch from single

standby mode to multiple standby mode by setting this parameter without shutting down the primary. The *host*:*port* pairs that you specify for the **hadr_target_list** configuration parameter for a primary determine which hosts act as standbys for that primary. The *host*:*port* pairs in the target list of a standby identify the standby hosts to be used if this standby takes over as the new HADR primary database.

The default setting of NULL, indicates that HADR runs in single standby mode.

As with the **hadr_remote_host** and **hadr_local_host** database configuration parameters, the host that you specify for the **hadr_target_list** configuration parameter can be either a host name or an IP address. A host name can contain alphanumerics, dashes, and underscores only. As with the **hadr_remote_svc** and **hadr_local_svc** configuration parameters, the port that you specify for the **hadr_target_list** configuration parameter can be either a port number or a service name. A service name can consist of any characters. A host name is mapped to an IP address, and a service name is mapped to a port number. The values that you specify for the **hadr_target_list** configuration parameter can contain a combination of host names, host IP addresses, service names, and port values. Specify the *host*:*port* pairs in the following format:

```
host1:port1|host2:port2|host3:port3
```

You must enclose Numerical Internet Protocol version 6 (IPv6) addresses in square brackets ([]) to differentiate the IP part from the port part, as in the following example:

```
[FEDC:BA98:7654:3210:FEDC:BA98:7654:3210]:4000
```

The first entry in the target list of an HADR database is designated as the *principal HADR standby database*, and any other entries are called *auxiliary HADR standby databases*. You can configure the principal standby database and the primary database to use any synchronization mode; however, the synchronization mode of the principal standby is determined by the setting for the **hadr_syncmode** configuration parameter on the primary. The synchronization mode of the auxiliary standby targets is always SUPERASYNC. When you start HADR on the primary, the values for the **hadr_remote_host** and **hadr_remote_svc** configuration parameters are automatically set to those of the principal standby unless you set the **DB2_HADR_NO_IP_CHECK** registry variable to ON.

There is no requirement for symmetry or reciprocal settings. For example, database A can designate database B as its principal standby, and database B can designate database C as its principal standby. However, role switch between any pair of the databases must be able to occur. For example, if database B is listed in the target list of database A, database A must be listed in the target list of database B.

Although the **hadr_target_list** parameter can be updated and take effect while the database is online, there are some restrictions if HADR is active:

- You cannot change the principal standby of the primary without first stopping HADR on the primary.
- You cannot remove a standby from the list if it is connected to the primary. To disconnect a standby, simply deactivate it. Then you can remove it from the primary's target list.
- You cannot dynamically update the **hadr_target_list** configuration parameter for a standby unless you have enabled the HADR reads on standby feature.
- You cannot remove the primary database from the target list of a standby if the standby is connected to the primary.

- The target list must contain IP address that are either IPv4 or IPv6, but not a combination of the two.

hadr_timeout - HADR timeout value

This parameter specifies the time (in seconds) that the high availability disaster recovery (HADR) process waits before considering a communication attempt to have failed.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients

Parameter type

- Configurable[14]

Default [range]
120 [1 - 4 294 967 295]

indexrec - Index re-creation time

This parameter indicates when the database manager attempts to rebuild invalid indexes, and whether or not any index build is redone during rollforward or high availability disaster recovery (HADR) log replay on the standby database.

Configuration type
Database and Database Manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]

UNIX Database Manager
`restart [restart; restart_no_redo; access; access_no_redo]`

Windows Database Manager
`restart [restart; restart_no_redo; access; access_no_redo]`

Database
Use system setting [`system; restart; restart_no_redo; access; access_no_redo`]

There are five possible settings for this parameter:

SYSTEM
Use system setting specified in the database manager configuration file to

14. Changes to this parameter take effect on database activation. If the database is already online, you can have changes take effect by stopping and restarting HADR on the primary database.

decide when invalid indexes are rebuilt, and whether any index build log records are to be redone during rollforward or HADR log replay.

Note: This setting is only valid for database configurations.

ACCESS
Invalid indexes are rebuilt when the underlying table is first accessed. Any fully logged index builds are redone during rollforward or HADR log replay. When HADR is started and an HADR takeover occurs, any invalid indexes are rebuilt after takeover when the underlying table is first accessed.

ACCESS_NO_REDO
Invalid indexes are rebuilt when the underlying table is first accessed. Any fully logged index build is not redone during rollforward or HADR log replay and those indexes are left invalid. When HADR is started and an HADR takeover takes place, any invalid indexes are rebuilt after takeover when the underlying table is first accessed. Access to the underlying tables on the new primary cause index rebuild, which causes log records to be written and then sent to the new standby, which in turn causes the indexes to be invalidated on the standby.

RESTART
The default value for **indexrec**. Invalid indexes are rebuilt when a **RESTART DATABASE** command is either explicitly or implicitly issued. Any fully logged index build will be redone during rollforward or HADR log replay. When HADR is started and an HADR takeover takes place, any invalid indexes will be rebuilt at the end of takeover.

Note: In a DB2 pureScale environment, indexes are rebuilt only during a group crash recovery, not as part of member crash recovery.

Note: When a database terminates abnormally while applications are connected to it, and the autorestart parameter is enabled, a **RESTART DATABASE** command is implicitly issued when an application connects to a database. If the command is not issued, the invalid indexes are rebuilt the next time the underlying table is accessed.

RESTART_NO_REDO
Invalid indexes are rebuilt when a **RESTART DATABASE** command is either explicitly or implicitly issued. Any fully logged index build is not redone during rollforward or HADR log replay and instead those indexes are rebuilt when rollforward completes or when HADR takeover takes place. Takeover causes index rebuild on underlying tables on the new primary, which causes log records to be written and then sent to the new standby, which in turn causes the indexes to be invalidated on the standby.

When a database terminates abnormally while applications are connected to it, and the autorestart parameter is enabled, a **RESTART DATABASE** command is implicitly issued when an application connects to a database. If the command is not issued, the invalid indexes are rebuilt the next time the underlying table is accessed.

Indexes can become invalid when fatal disk problems occur. If this happens to the data itself, the data could be lost. However, if this happens to an index, the index can be recovered by recreating it. If an index is rebuilt while users are connected to the database, two problems could occur:

- An unexpected degradation in response time might occur as the index file is re-created. Users accessing the table and using this particular index would wait while the index was being rebuilt.
- Unexpected locks might be held after index recreation, especially if the user transaction that caused the index to be re-created never performed a COMMIT or ROLLBACK.

Recommendation: The best choice for this option on a high-user server and if restart time is not a concern, would be to have the index rebuilt at **DATABASE RESTART** time as part of the process of bringing the database back online after a crash.

Setting this parameter to ACCESS or to ACCESS_NO_REDO result in a degradation of the performance of the database manager while the index is being re-created. Any user accessing that specific index or table would have to wait until the index is re-created.

If this parameter is set to RESTART, the time taken to restart the database is longer due to index re-creation, but normal processing would not be impacted once the database has been brought back online.

Note: At database recovery time, all SQL procedure executables on the file system that belong to the database being recovered are removed. If **indexrec** is set to RESTART, all SQL procedure executables are extracted from the database catalog and put back on the file system at the next connection to the database. If **indexrec** is not set to RESTART, an SQL executable is extracted to the file system only on first execution of that SQL procedure.

The difference between the RESTART and the RESTART_NO_REDO values, or between the ACCESS and the ACCESS_NO_REDO values, is only significant when full logging is activated for index build operations, such as **CREATE INDEX** and **REORG INDEX** operations, or for an index rebuild. You can activate logging by enabling the **logindexbuild** database configuration parameter or by enabling LOG INDEX BUILD when altering a table. By setting **indexrec** to either RESTART or ACCESS, operations involving a logged index build can be rolled forward without leaving the index object in an invalid state, which would require the index to be rebuilt at a later time.

jdk_64_path - 64-Bit Software Developer's Kit for Java installation path DAS

This parameter specifies the directory under which the 64-Bit Software Developer's Kit (SDK) for Java, to be used for running DB2 administration server functions, is installed.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
Null [any valid path]

Note: This is different from the **jdk_path** configuration parameter, which specifies a 32-bit SDK for Java.

Environment variables used by the Java interpreter are computed from the value of this parameter. This parameter is only used on those platforms that support both 32- and 64-bit instances.

Those platforms are:

- 64-bit kernels of AIX, HP-UX, and Solaris operating systems
- 64-bit Windows on x64 and IPF
- 64-bit Linux kernel on AMD64 and Intel EM64T systems (x64), POWER, and zSeries.

On all other platforms, only **jdk_path** is used.

Because there is no default value for this parameter, you should specify a value when you install the SDK for Java.

This parameter can only be updated from a Version 8 command line processor (CLP).

locklist - Maximum storage for lock list

This parameter indicates the amount of storage that is allocated to the lock list. There is one lock list per database and it contains the locks held by all applications concurrently connected to the database.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
`Automatic [4 - 134217728]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
When the first application connects to the database

When freed
When last application disconnects from the database

Locking is the mechanism that the database manager uses to control concurrent access to data in the database by multiple applications. Both rows and tables can be locked. The database manager can also acquire locks for internal use.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active

Although the value of **locklist** can be tuned together with the **maxlocks** parameter, disabling self tuning of the **locklist** parameter does not automatically disable self tuning of the **maxlocks** parameter. Enabling self tuning of the **locklist** parameter automatically enables self tuning of the **maxlocks** parameter.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the **self_tuning_mem** configuration parameter is set to ON.)

On all platforms, each lock requires 128 or 256 bytes of the lock list, depending on whether other locks are held on the object:

- 256 bytes are required to hold a lock on an object that has no other locks held on it
- 128 bytes are required to record a lock on an object that has an existing lock held on it.

When the percentage of the lock list used by one application reaches **maxlocks**, the database manager will perform lock escalation, from row to table, for the locks held by the application. This calculation is an approximation, assuming shared locks only. The percentage of the lock list used is calculated by multiplying the number of locks held by the application by the value required to hold a lock on an object that has other locks held on it. Although the escalation process itself does not take much time, locking entire tables (versus individual rows) decreases concurrency, and overall database performance might decrease for subsequent accesses against the affected tables. Suggestions of how to control the size of the lock list are:

- Perform frequent COMMITs to release locks.
- When performing many updates, lock the entire table before updating (using the SQL LOCK TABLE statement). This will use only one lock, keeps others from interfering with the updates, but does reduce concurrency of the data.

  You can also use the LOCKSIZE option of the ALTER TABLE statement to control how locking is done for a specific table.

  Use of the Repeatable Read isolation level might result in an automatic table lock.
- Use the Cursor Stability isolation level when possible to decrease the number of share locks held. If application integrity requirements are not compromised use Uncommitted Read instead of Cursor Stability to further decrease the amount of locking.
- Set **locklist** to AUTOMATIC. The lock list will increase synchronously to avoid lock escalation or a lock list full situation.

Once the lock list is full, performance can degrade since lock escalation will generate more table locks and fewer row locks, thus reducing concurrency on shared objects in the database. Additionally there might be more deadlocks between applications (since they are all waiting on a limited number of table locks), which will result in transactions being rolled back. Your application will receive an SQLCODE of -912 when the maximum number of lock requests has been reached for the database.

Recommendation: If lock escalations are causing performance concerns you might need to increase the value of this parameter or the **maxlocks** parameter. You can use the database system monitor to determine if lock escalations are occurring. Refer to the **lock_escals** (lock escalations) monitor element.

The following steps might help in determining the number of pages required for your lock list:

1. Calculate a lower bound for the size of your lock list, using *one* of the following calculations, depending on your environment:
   a.
   ```
   (512 * 128 * maxappls) / 4096
   ```
   b. with Concentrator enabled:
   ```
   (512 * 128 * max_coordagents) / 4096
   ```
   c. in a partitioned database with Concentrator enabled:
   ```
   (512 * 128 * max_coordagents * number of database partitions) / 4096
   ```
   where 512 is an estimate of the average number of locks per application and 128 is the number of bytes required for each lock against an object that has an existing lock.
2. Calculate an upper bound for the size of your lock list:
   ```
   (512 * 256 * maxappls) / 4096
   ```
   where 256 is the number of bytes required for the first lock against an object.

   Note: While in a DB2 pureScale environment, set the **locklist** configuration parameter to be equal to the upper bound of the value calculated in this step and 3% of the total number of pages for all of the buffer pools existing in the currently connected database.
3. Estimate the amount of concurrency you will have against your data and based on your expectations, choose an initial value for **locklist** that falls between the upper and lower bounds that you have calculated.
4. Using the database system monitor, as described in the following paragraph, tune the value of this parameter.

If **maxappls** or **max_coordagents** are set to AUTOMATIC in your applicable scenario, you should also set **locklist** to AUTOMATIC.

You can use the database system monitor to determine the maximum number of locks held by a given transaction. Refer to the **locks_held_top** (maximum number of locks held) monitor element.

This information can help you validate or adjust the estimated number of locks per application. In order to perform this validation, you will have to sample several applications, noting that the monitor information is provided at a transaction level, not an application level.

You might also want to increase **locklist** if **maxappls** is increased, or if the applications being run perform infrequent commits.

You should consider rebinding applications (using the **REBIND** command) after changing this parameter.

locktimeout - Lock timeout

This parameter specifies the number of seconds that an application will wait to obtain a lock, helping avoid global deadlocks for applications.

Configuration type
Database

Parameter type
- Configurable

Default [range]
`-1 [-1; 0 - 32 767 ]`

Unit of measure
Seconds

If you set this parameter to `0`, locks are not waited for. In this situation, if no lock is available at the time of the request, the application immediately receives a -911.

If you set this parameter to `-1`, lock timeout detection is turned off. In this situation a lock will be waited for (if one is not available at the time of the request) until either of the following events occur:
- The lock is granted
- A deadlock occurs.

Note: The value you specify for this configuration parameter is not used to control lock timeouts for event monitor target tables. Event monitors use a separate, non-configurable setting that will cause locks on event monitor tables to time out.

Recommendation: In a transaction processing (OLTP) environment, you can use an initial starting value of 30 seconds. In a query-only environment you could start with a higher value. In both cases, you should use benchmarking techniques to tune this parameter.

The value should be set to quickly detect waits that are occurring because of an abnormal situation, such as a transaction that is stalled (possibly as a result of a user leaving their workstation). You should set it high enough so valid lock requests do not time out because of peak workloads, during which time, there is more waiting for locks.

You can use the database system monitor to help you track the number of times an application (connection) experienced a lock timeout or that a database detected a timeout situation for all applications that were connected.

High values of the **`lock_timeout`** (number of lock timeouts) monitor element can be caused by:
- Too low a value for this configuration parameter.
- An application (transaction) that is holding locks for an extended period. You can use the database system monitor to further investigate these applications.
- A concurrency problem, that could be caused by lock escalations (from row-level to a table-level lock).

log_appl_info - Application information log record database configuration parameter

This parameter specifies that the application information log record is written at the start of each update transaction.

Configuration type
Database

Parameter type
Configurable

Default [range]
No [Yes; No]

When set to YES, an extra log record is added at the start of each update transaction. The application information log record is used for the transaction (refer to Log record header).

Enable the **log_appl_info** configuration parameter if there are tables created with the DATA CAPTURE CHANGES option for replication purpose (or other purpose such as auditing).

log_ddl_stmts - Log Data Definition Language (DDL) statements database configuration parameter

This parameter specifies that extra information regarding Data Definition Language (DDL) statements will be written to the log.

Configuration type
Database

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
No [Yes; No]

The default setting of the **log_ddl_stmts** configuration parameter avoids the overhead of writing extra log records for DDL operations. Set this parameter to YES if you need to replicate DDL operations and a replication capture program is used to capture changes from the log.

log_retain_status - Log retain status indicator

If the **logarchmeth1** database configuration parameter is set to **logretain** then the log retain status parameter will show a value of RECOVERY, otherwise it will show a value of NO.

Configuration type
Database

Parameter type
Informational

logarchcompr1 - Primary archived log file compression configuration parameter

This parameter specifies whether the log files written to the primary archive destination for logs are compressed.

Configuration type
Database

Applies to

- Database servers with local and remote clients
- Clients

- Database servers with local clients
- Partitioned database servers with local and remote clients

Parameter type
Configurable online

Default [range]
OFF [ON]

OFF Specifies that log files written to the primary archive destination for logs are not compressed.

ON Specifies that log files written to the primary archive destination for logs are compressed. If set dynamically, log files already archived are not compressed.

Note:

- If you set the **logarchmeth1** configuration parameter to a value other than DISK, TSM, or VENDOR, log archive compression has no effect regardless of the **logarchcompr1** configuration parameter setting.

logarchcompr2 - Secondary archived log file compression configuration parameter

This parameter specifies whether the log files written to the secondary archive destination for logs are compressed.

Configuration type
Database

Applies to

- Database servers with local and remote clients
- Clients
- Database servers with local clients
- Partitioned database servers with local and remote clients

Parameter type
Configurable online

Default [range]
OFF [ON]

OFF Specifies that log files written to the secondary archive destination for logs are not compressed.

ON Specifies that log files written to the secondary archive destination for logs are compressed. If set dynamically, log files already archived are not compressed.

Note:

- If you set the **logarchmeth2** configuration parameter to a value other than DISK, TSM, or VENDOR, log archive compression has no effect regardless of the **logarchcompr2** configuration parameter setting.

logarchmeth1 - Primary log archive method

This parameter specifies the media type of the primary destination for logs that are archived from the current log path.

Configuration type
Database

Applies to

- Database servers with local and remote clients
- Clients
- Database servers with local clients
- Partitioned database servers with local and remote clients

Parameter type
Configurable online

Default [range]
OFF [LOGRETAIN, USEREXIT, DISK, TSM, VENDOR]

OFF Specifies that the log archiving method is not used. If you set both the **logarchmeth1** and **logarchmeth2** configuration parameters to OFF, which is the default, the database is considered to be using circular logging and is not rollforward recoverable.

LOGRETAIN
Specifies that active log files are retained and become online archive log files for use in rollforward recovery.

USEREXIT
Specifies that log retention logging is performed and that a user exit program should be used to archive and retrieve the log files. Log files are archived when they are full. They are retrieved when the **ROLLFORWARD** utility must use them to restore a database.

DISK You must follow this value with a colon (:) and then a fully qualified existing path name where the log files will be archived. For example, if you set the **logarchmeth1** configuration parameter to DISK:/u/dbuser/archived_logs, the archive log files are placed in the /u/dbuser/archived_logs/*instance*/*dbname*/*nodename*/*logstream*/*chainid*/ directory.

Note: If you are archiving to tape, you can use the **db2tapemgr** utility to store and retrieve log files.

TSM If specified without any additional configuration parameters, indicates that log files are archived on the local TSM server by using the default management class. If you follow this option with a colon (:) and a TSM management class, the log files are archived using the specified management class.

When archiving logs using TSM, before using the management class that is specified by the database configuration parameter, TSM attempts to bind the object to the management class that you specified in the INCLUDE-EXCLUDE list in the TSM client options file. If a match is not found, the default TSM management class that you specified on the TSM server is used. TSM then rebinds the object to the management class that you specified for the database configuration parameter. Thus, the default management class and the management class that you specify for the database configuration parameter must contain an archive copy group, or the archive operation fails. Examples of TSM entries are:

- With a management class specified: `db2 update db cfg for mydb using logarchmeth1 TSM:DB2_LOGS`

- With no management class specified: `db2 update db cfg for mydb using logarchmeth1 TSM`

VENDOR Specifies that a vendor library is used to archive the log files. You must follow this value with a colon (:) and the name of the library. The APIs in the library must use the backup and restore APIs for vendor products. An example of a vendor entry is: `db2 update db cfg for mydb using logarchmeth1 VENDOR:/home/dbuser/vendorLib/<library name>`

Notes:

- If you set either the **logarchmeth1** or **logarchmeth2** configuration parameter to a value other than OFF, the database is configured for rollforward recovery.
- If you use the `userexit` or `logretain` option for the **logarchmeth1** configuration parameter, you must set the **logarchmeth2** configuration parameter to OFF.
- To assign an archive path with a space in it, use the **db2CfgSet** API.

logarchmeth2 - Secondary log archive method

This parameter specifies the media type of the secondary destination for logs that are archived from either the current log path or the mirror log path.

Configuration type
Database

Applies to

- Database servers with local and remote clients
- Clients
- Database servers with local clients
- Partitioned database servers with local and remote clients

Parameter type
Configurable online

Default [range]
OFF [DISK, TSM, VENDOR]

OFF Specifies that the log archiving method is not used. If you set both **logarchmeth1** and **logarchmeth2** configuration parameters to OFF, the database is considered to be using circular logging and is not rollforward recoverable. This is the default.

DISK You must follow this value with a colon (:) and then a fully qualified existing path name where the log files will be archived. For example, if you set the **logarchmeth1** configuration parameter to `DISK:/u/dbuser/archived_logs` the archive log files are placed in the `/u/dbuser/archived_logs/`*instance*`/`*dbname*`/`*nodename*`/`*chainid*`/` directory.

Note: If you are archiving to tape, you can use the **db2tapemgr** utility to store and retrieve log files.

TSM If specified without any additional configuration parameters, indicates that log files are archived on the local TSM server by using the default management class. If you follow this option with

a colon (:) and a TSM management class, the log files are archived using the specified management class.

When archiving logs using TSM, before using the management class that is specified by the database configuration parameter, TSM attempts to bind the object to the management class that you specified in the INCLUDE-EXCLUDE list in the TSM client options file. If a match is not found, the default TSM management class that you specified on the TSM server is used. TSM then rebinds the object to the management class that you specified for the database configuration parameter. Thus, the default management class and the management class that you specify for the database configuration parameter must contain an archive copy group, or the archive operation fails.

VENDOR
Specifies that a vendor library is used to archive the log files. You must follow this value with a colon (:) and the name of the library. The APIs in the library must use the backup and restore APIs for vendor products.

Notes:

1. If you set either the **logarchmeth1** or **logarchmeth2** configuration parameter to a value other than OFF, the database is configured for rollforward recovery.
2. If you use the **userexit** or **logretain** option for the **logarchmeth1** configuration parameter, you must set the **logarchmeth2** configuration parameter to OFF.

If you use the **logarchmeth2** configuration parameter to specify a path, log files will be archived to both this destination and the destination that is specified by the **logarchmeth1** database configuration parameter. Which log files the **logarchmeth2** configuration parameter archives depends on whether you also set the value of the **mirrorlogpath** database configuration parameter:

- If you do not set the value of the **mirrorlogpath** configuration parameter, the **logarchmeth2** configuration parameter archives log files from the current log path, as specified by the **logpath** configuration parameter.
- If you set the value of the **mirrorlogpath** configuration parameter, the **logarchmeth2** configuration parameter archives log files from the mirror log path.

logarchopt1 - Primary log archive options

This parameter specifies the options field for the primary destination for archived logs (if required).

Configuration type
Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Default [range]
Null [not applicable]

Restrictions
In TSM environments configured to support proxy nodes, the `"-fromnode=nodename"` option and the `"-fromowner=ownername"` option are not compatible with the `"-asnodename=nodename"` option and cannot be used together. Use the `-asnodename` option for TSM configurations using proxy nodes and the other two options for other types of TSM configurations. For more information, see "Configuring a Tivoli Storage Manager client".

logarchopt2 - Secondary log archive options

This parameter specifies the options field for the secondary destination for archived logs (if required).

Configuration type
Database

Applies to
- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Default [range]
Null [not applicable]

Restrictions
In TSM environments configured to support proxy nodes, the `"-fromnode=nodename"` option and the `"-fromowner=ownername"` option are not compatible with the `"-asnodename=nodename"` option and cannot be used together. Use the `-asnodename` option for TSM configurations using proxy nodes and the other two options for other types of TSM configurations. For more information, see "Configuring a Tivoli Storage Manager client".

logbufsz - Log buffer size

This parameter allows you to specify the amount of the database heap (defined by the **dbheap** parameter) to use as a buffer for log records before writing these records to disk.

Configuration type
Database

Parameter type
- Configurable
- Configurable by member in a DB2 pureScale environment

Default [range]

32-bit platforms
`256 [4 - 4 096 ]`

64-bit platforms
`256 [4 - 131 070 ]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

Log records are written to disk when one of the following occurs:

- A transaction commits or a group of transactions commit, as defined by the **mincommit** configuration parameter
- The log buffer is full
- As a result of some other internal database manager event.

This parameter must also be less than or equal to the **dbheap** parameter. Buffering the log records will result in more efficient logging file I/O because the log records will be written to disk less frequently and more log records will be written at each time.

Recommendation: Increase the size of this buffer area if there is considerable read activity on a dedicated log disk, or there is high disk utilization. When increasing the value of this parameter, you should also consider the **dbheap** parameter since the log buffer area uses space controlled by the **dbheap** parameter.

You can use the database system monitor to determine how often the log buffer has become full, requiring it to be written to disk before new log records can be written. Refer to the **num_log_buffer_full** (number of full log buffers) monitor element.

logfilsiz - Size of log files

This parameter defines the size of each primary and secondary log file. The size of these log files limits the number of log records that can be written to them before they become full and a new log file is required.

Configuration type
Database

Parameter type
Configurable

Default [range]

UNIX `1000 [4 - 1 048 572]`

Windows
`1000 [4 - 1 048 572]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

The use of primary and secondary log files as well as the action taken when a log file becomes full are dependent on the type of logging that is being performed:

- Circular logging

  A primary log file can be reused when the changes recorded in it have been committed. If the log file size is small and applications have processed a large number of changes to the database without committing the changes, a primary

log file can quickly become full. If all primary log files become full, the database manager will allocate secondary log files to hold the new log records.

- Log retain or archive logging

  When a log file is full, it is closed and not overwritten. If archive logging is configured then the log file will be subsequently archived.

Recommendation: You must balance the size of the log files with the number of log files:

- The value of the **logfilsiz** should be increased if the database has a large number of update, delete, or insert transactions running against it which will cause the log file to become full very quickly.

  Note: The upper limit of log file size, combined with the upper limit of the number of log files (**logprimary** + **logsecond**), gives an upper limit of 1024 GB of active log space.

  A log file that is too small can affect system performance because of the processing time of archiving old log files, allocating new log files, and waiting for a usable log file.

- The value of the **logfilsiz** should be reduced if disk space is scarce, since primary logs are preallocated at this size.

  A log file that is too large can reduce your flexibility when managing archived log files and copies of log files, since some media might not be able to hold an entire log file.

  Also, the size of the log file can impact the frequency of which log files are archived and the amount of time it takes to archive an individual log file. These factors will impact the availability of log files in the archive location for disaster recovery scenarios if something happens to the primary server. The **ARCHIVE LOG FOR DATABASE** command can be used to truncate and archive log files on a more frequent basis if necessary.

If you are using log retention, the current active log file is closed and truncated when the last application disconnects from a database. When the next connection to the database occurs, the next log file is used. Therefore, if you understand the logging requirements of your concurrent applications, you might be able to determine a log file size that will not allocate excessive amounts of wasted space.

loghead - First active log file

This parameter contains the name of the log file that is currently active.

Configuration type
Database

Parameter type
Informational

logindexbuild - Log index pages created

This parameter specifies whether index creation, recreation, or reorganization operations are to be logged so that indexes can be reconstructed during DB2 rollforward operations or high availability disaster recovery (HADR) log replay procedures.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Default [range]

Off [On; Off]

logpath - Location of log files

This parameter contains the current path being used for logging purposes.

Configuration type

Database

Parameter type

Informational

You cannot change this parameter directly, because it is set by the database manager after a change to the **newlogpath** parameter takes effect.

The default log path for a member is a directory within the global database directory. For example, given an instance name of dbinst, a database name of dbname, and a user named dbuser, the default log path for a member might be `/home/dbuser/dbinst/NODE0000/SQL00001/LOGSTREAM0000`. In a DB2 pureScale environment, there is one LOGSTREAM*xxxx* directory for each member.

logprimary - Number of primary log files

This parameter allows you to specify the number of primary log files to be preallocated. The primary log files establish a fixed amount of storage allocated to the recovery log files.

Configuration type

Database

Parameter type

Configurable

Default [range]

`3 [ 2 - 256 ]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure

Counter

When allocated

- The database is created
- A log is moved to a different location (which occurs when the **newlogpath** parameter is updated)
- When the database is next started following an increase in the value of this parameter (**logprimary**), provided that the database is not started as an HADR standby database

- A log file becomes inactive and a new log file is required to keep at least **logprimary** logs in the primary log path (the **logarchmeth1** or **logarchmeth2** parameter must not be set to a value other than OFF).
- If the **logfilsiz** parameter has been changed, the log files are re-sized during the next database startup, provided that it is not started as an HADR standby database

When freed

Not freed unless this parameter decreases. If decreased, unneeded log files are deleted during the next connection to the database.

Under circular logging, the primary logs are used repeatedly in sequence. That is, when a log is full, the next primary log in the sequence is used if it is available. A log is considered available if all units of work with log records in it have been committed or rolled-back. If the next primary log in sequence is not available, then a secondary log is allocated and used. Additional secondary logs are allocated and used until the next primary log in the sequence becomes available or the limit imposed by the **logsecond** parameter is reached. These secondary log files are dynamically deallocated as they are no longer needed by the database manager.

The number of primary and secondary log files must comply with the following:

- If **logsecond** has a value of -1, **logprimary** <= 256.
- If **logsecond** does not have a value of -1, (**logprimary** + **logsecond**) <= 256.

Recommendation: The value chosen for this parameter depends on a number of factors, including the type of logging being used, the size of the log files, and the type of processing environment (for example, length of transactions and frequency of commits).

Increasing this value will increase the disk requirements for the logs because the primary log files are preallocated during the very first connection to the database.

If you find that secondary log files are frequently being allocated, you might be able to improve system performance by increasing the log file size (**logfilsiz**) or by increasing the number of primary log files.

For databases that are not frequently accessed, in order to save disk storage, set the parameter to 2. For databases enabled for rollforward recovery, set the parameter larger to avoid the additional processing time of allocating new logs almost immediately.

You can use the database system monitor to help you size the primary log files. Observation of the following monitor values over a period of time will aid in better tuning decisions, as average values might be more representative of your ongoing requirements.

- **sec_log_used_top** (maximum secondary log space used)
- **tot_log_used_top** (maximum total log space used)
- **sec_logs_allocated** (secondary logs allocated currently)

logsecond - Number of secondary log files

This parameter specifies the number of secondary log files that are created and used for recovery log files. The secondary log files are created only as needed.

Configuration type

Database

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
10 [-1; 0 - 254]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Counter

When allocated
As needed when **logprimary** is insufficient. For more allocation details, see the details that follow.

When freed
Over time as the database manager determines which secondary log files are no longer needed.

When the primary log files become full, the secondary log files (of size **logfilsiz**) are allocated one at a time as needed, up to a maximum number as controlled by this parameter. If more secondary log files are required than are allowed by this parameter, an error code will be returned to the application.

If you set **logsecond** to -1, the database is configured with infinite active log space. There is no limit on the size or the number of in-flight transactions running on the database. If you set **logsecond** to -1, you still use the **logprimary** and **logfilsiz** configuration parameters to specify how many log files the database manager should keep in the active log path. If the database manager needs to read log data from a log file, but the file is not in the active log path, the database manager retrieves the log file from the archive to the active log path. (The database manager retrieves the files to the overflow log path, if you have configured one.) Once the log file is retrieved, the database manager will cache this file in the active log path so that other reads of log data from the same file will be fast. The database manager will manage the retrieval, caching, and removal of these log files as required.

Note: You cannot configure infinite active log space in DB2 pureScale environments.

If your log path is a raw device, you must configure the **overflowlogpath** configuration parameter in order to set **logsecond** to -1.

By setting **logsecond** to -1, you will have no limit on the size of the unit of work or the number of concurrent units of work. However, rollback (both at the savepoint level and at the unit of work level) could be very slow due to the need to retrieve log files from the archive. Crash recovery could also be very slow for the same reason. The database manager writes a message to the administration notification log to warn you that the current set of active units of work has exceeded the primary log files. This is an indication that rollback or crash recovery could be extremely slow.

To set **logsecond** to -1, the **logarchmeth1** configuration parameter must be set to a value other than OFF or LOGRETAIN.

Recommendation: Use secondary log files for databases that have periodic needs for large amounts of log space. For example, an application that is run once a month might require log space beyond that provided by the primary log files. Because secondary log files do not require permanent file space, they are advantageous in this type of situation.

When infinite logging is enabled (**logsecond** to -1), the database manager does not reserve active log space for transactions that may need to roll back and write log records. During rollback processing, if both the active log path and archive target are full (or if the archive target is inaccessible), then the **blk_log_dsk_ful** (block on log disk full db configuration parameter) should also be ENABLED to avoid database failures.

max_log - Maximum log per transaction

This parameter specifies if there is a limit to the percentage of the primary log space that a transaction can consume, and what that limit is.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
0 [0 - 100]

Unit of measure
Percentage

If the value is not 0, this parameter indicates the percentage of primary log space that can be consumed by one transaction.

If the value is set to 0, there is no limit to the percentage of total primary log space that a transaction can consume.

If an application violates the **max_log** configuration, the application is forced to disconnect from the database and the transaction is rolled back.

You can override this behavior by setting the *DB2_FORCE_APP_ON_MAX_LOG* registry variable to "FALSE". This will cause transactions that violate the **max_log** configuration to fail; however, the application can still commit the work completed by previous statements in the unit or work, or it can roll the completed work back to undo the unit of work.

This parameter, along with the **num_log_span** configuration parameter, can be useful when enabling infinite active logspace. If infinite logging is on (that is, if *logsecond* is set to -1) then transactions are not restricted to the upper limit of the number of log files (*logprimary* + *logsecond*). When the value of *logprimary* is reached, DB2 starts to archive the active logs, rather than failing the transaction. This can cause problems if, for example, an application contains a long running transaction that is left uncommitted. If this occurs, the active logspace continues to

grow, possibly leading to poor crash recovery performance. To prevent this, you can specify values for one or both of the **max_log** or **num_log_span** configuration parameters.

Note: The following DB2 commands are excluded from the limitation imposed by the **max_log** configuration parameter: ARCHIVE LOG, BACKUP DATABASE, LOAD, REORG, RESTORE DATABASE, and ROLLFORWARD DATABASE.

maxappls - Maximum number of active applications

This parameter specifies the maximum number of concurrent applications that can be connected (both local and remote) to a database. Since each application that attaches to a database causes some private memory to be allocated, allowing a larger number of concurrent applications will potentially use more memory.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
`Automatic [1 - 60 000]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Counter

Setting **maxappls** to `automatic` has the effect of allowing any number of connected applications. The database manager will dynamically allocate the resources it needs to support new applications.

If you do not want to set this parameter to `automatic`, the value of this parameter must be equal to or greater than the sum of the connected applications, plus the number of these same applications that might be concurrently in the process of completing a two-phase commit or rollback. Then add to this sum the anticipated number of indoubt transactions that might exist at any one time.

When an application attempts to connect to a database, but **maxappls** has already been reached, an error is returned to the application indicating that the maximum number of applications have been connected to the database.

In a partitioned database environment, this is the maximum number of applications that can be concurrently active against a database partition. This parameter limits the number of active applications against the database partition on a database partition server, regardless of whether the server is the coordinator node for the application or not. The catalog node in a partitioned database environment requires a higher value for **maxappls** than is the case for other types of environments because, in the partitioned database environment, every application requires a connection to the catalog node.

Recommendation: Increasing the value of this parameter without lowering the **maxlocks** parameter or increasing the **locklist** parameter could cause you to reach the database limit on locks (**locklist**) rather than the application limit and as a result cause pervasive lock escalation problems.

To a certain extent, the maximum number of applications is also governed by **max_coordagents**. An application can only connect to the database, if there is an available connection (**maxappls**) as well as an available coordinating agent (**max_coordagents**).

maxfilop - Maximum database files open per database

This parameter specifies the maximum number of file handles that can be open per application.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Transaction boundary

Default [range]

AIX, Sun, HP, and Linux 64-bit
`61 440 [64 - 61 440]`

Linux 32-bit
`30 720 [64 - 30 720]`

Windows 32-bit
`32 768 [64 - 32 768]`

Windows 64-bit
`65 335 [64 - 65 335]`

Unit of measure
Counter

If opening a file causes this value to be exceeded, some files in use by this database are closed. If **maxfilop** is too small, the additional processing time of opening and closing files will become excessive and might degrade performance.

Both SMS table spaces and DMS table space file containers are treated as files in the database manager's interaction with the operating system, and file handles are required. More files are generally used by SMS table spaces compared to the number of containers used for a DMS file table space. Therefore, if you are using SMS table spaces, you will need a larger value for this parameter compared to what you would require for DMS file table spaces.

You can also use this parameter to ensure that the overall total of file handles used by the database manager does not exceed the operating system limit by limiting the number of handles per database to a specific number; the actual number will vary depending on the number of databases running concurrently.

maxlocks - Maximum percent of lock list before escalation

This parameter defines a percentage of the lock list held by an application that must be filled before the database manager performs lock escalation.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
`Automatic [1 - 100 ]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Percentage

Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list. When the number of locks held by any one application reaches this percentage of the total lock list size, lock escalation will occur for the locks held by that application. Lock escalation also occurs if the lock list runs out of space.

The database manager determines which locks to escalate by looking through the lock list for the application and finding the table with the most row locks. If after replacing these with a single table lock, the **maxlocks** value is no longer exceeded, lock escalation will stop. If not, it will continue until the percentage of the lock list held is below the value of **maxlocks**. The **maxlocks** parameter multiplied by the **maxappls** parameter cannot be less than 100.

When this parameter is set to `AUTOMATIC`, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active.

Although the value of **locklist** can be tuned together with the **maxlocks** parameter, disabling self tuning of the **locklist** parameter does not automatically disable self tuning of the **maxlocks** parameter. Enabling self tuning of the **locklist** parameter automatically enables self tuning of the **maxlocks** parameter.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the **self_tuning_mem** configuration parameter is set to `ON`).

On all platforms, each lock requires 128 or 256 bytes of the lock list, depending on whether other locks are held on the object:

- 256 bytes are required to hold a lock on an object that has no other locks held on it.

- 128 bytes are required to record a lock on an object that has an existing lock held on it.

Recommendation: The following formula allows you to set **maxlocks** to allow an application to hold twice the average number of locks:

```
maxlocks = 2 * 100 / maxappls
```

Where 2 is used to achieve twice the average and 100 represents the largest percentage value allowed. If you have only a few applications that run concurrently, you could use the following formula as an alternative to the first formula:

```
maxlocks = 2 * 100 / (average number of applications running
concurrently)
```

One of the considerations when setting **maxlocks** is to use it in conjunction with the size of the lock list (**locklist**). The actual limit of the number of locks held by an application before lock escalation occurs is:

- `maxlocks * locklist * 4096 /(100 * 128)`

Where 4096 is the number of bytes in a page, 100 is the largest percentage value allowed for **maxlocks**, and 128 is the number of bytes per lock. If you know that one of your applications requires 1000 locks, and you do not want lock escalation to occur, then you should choose values for **maxlocks** and **locklist** in this formula so that the result is greater than 1000. (Using 10 for **maxlocks** and 100 for **locklist**, this formula results in greater than the 1000 locks needed.)

If **maxlocks** is set too low, lock escalation happens when there is still enough lock space for other concurrent applications. If **maxlocks** is set too high, a few applications can consume most of the lock space, and other applications will have to perform lock escalation. The need for lock escalation in this case results in poor concurrency.

You can use the database system monitor to help you track and tune this configuration parameter.

min_dec_div_3 - Decimal division scale to 3

This parameter is provided as a quick way to enable a change to computation of the scale for decimal division in SQL.

Configuration type
Database

Parameter type
Configurable

Default [range]
No [Yes, No]

The **min_dec_div_3** database configuration parameter changes the resulting scale of a decimal arithmetic operation involving division. It can be set to "Yes" or "No". The default value for **min_dec_div_3** is "No". If the value is "No", the scale is calculated as 31-p+s-s'. If set to "Yes", the scale is calculated as MAX(3, 31-p+s-s'). This causes the result of decimal division to always have a scale of at least 3. Precision is always 31.

Changing this database configuration parameter might cause changes to applications for existing databases. This can occur when the resulting scale for

decimal division would be impacted by changing this database configuration parameter. The following list shows some possible scenarios that might impact applications. These scenarios should be considered before changing the **min_dec_div_3** on a database server with existing databases.

- A static package will not change behavior until the package is rebound, either implicitly or explicitly. For example, after changing the value from NO to YES, the additional scale digits might not be included in the results until rebind occurs. For any changed static packages, an explicit **REBIND** command can be used to force a rebind.
- A check constraint involving decimal division might restrict some values that were previously accepted. Such rows now violate the constraint but will not be detected until one of the columns involved in the check constraint row is updated or the SET INTEGRITY statement with the IMMEDIATE CHECKED option is processed. To force checking of such a constraint, perform an ALTER TABLE statement in order to drop the check constraint and then perform an ALTER TABLE statement to add the constraint again.

Note: **min_dec_div_3** also has the following limitations:

1. The command **GET DB CFG FOR DBNAME** will not display the **min_dec_div_3** setting. The best way to determine the current setting is to observe the side-effect of a decimal division result. For example, consider the following statement:

   ```
   VALUES (DEC(1,31,0)/DEC(1,31,5))
   ```

   If this statement returns sqlcode SQL0419N, the database does not have **min_dec_div_3** support, or it is set to "No". If the statement returns 1.000, **min_dec_div_3** is set to "Yes".
2. **min_dec_div_3** does not appear in the list of configuration keywords when you run the following command: ? UPDATE DB CFG

mincommit - Number of commits to group

This parameter allows you to delay the writing of log records to disk until a minimum number of commits have been performed, helping to reduce the processing time the database manager requires when writing log records.

Important: This parameter is deprecated in Version 10.1 and might be removed in a future release. This parameter can still be used in releases before Version 10.1. In Version 10.1 and later releases, the value specified for this configuration parameter is ignored.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
1 [1 - 25]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Counter

This delay might improve performance when you have multiple applications running against a database and many commits are requested by the applications within a very short time frame.

This grouping of commits will only occur when the value of this parameter is greater than one and when the number of applications connected to the database is greater than or equal to the value of this parameter. When commit grouping is being performed, application commit requests could be held until either one second has elapsed or the number of commit requests equals the value of this parameter.

This parameter should be incremented by small amounts only; for example one (1). You should also use multi-user tests to verify that increasing the value of this parameter provides the expected results. Setting this parameter too high can negatively impact application response time.

Changes to the value specified for this parameter take effect immediately; you do not have to wait until all applications disconnect from the database.

Recommendation: It is recommended that this parameter is set to the default of 1.

You could sample the number of transactions per second and adjust this parameter to accommodate the peak number of transactions per second (or some large percentage of it). Accommodating peak activity would minimize the processing time of writing log records during transaction intensive periods.

If you increase **mincommit**, you might also need to increase the **logbufsz** parameter to avoid having a full log buffer force a write during these transaction intensive periods. In this case, the **logbufsz** should be equal to:

```
mincommit * (log space used, on average, by a transaction)
```

You can use the database system monitor to help you tune this parameter in the following ways:

- Calculating the peak number of transactions per second:

  Taking monitor samples throughout a typical day, you can determine your transaction intensive periods. You can calculate the total transactions by adding the following monitor elements:

  – **commit_sql_stmts** (commit statements attempted)
  – **rollback_sql_stmts** (rollback statements attempted)

  Using this information and the available timestamps, you can calculate the number of transactions per second.
- Calculating the log space used per transaction:

  Using sampling techniques over a period of time and a number of transactions, you can calculate an average of the log space used with the following monitor element:

  – **log_space_used** (unit of work log space used)

mirrorlogpath - Mirror log path

This parameter specifies a string of up to 242 bytes for the mirror log path. The string must point to a fully qualified path name.

Configuration type
Database

Parameter type
Configurable

Default [range]
Null [any valid path or device]

In both DB2 pureScale environments and DB2 Enterprise Server Edition environments, the database partition number and a log stream ID are automatically appended to the path, for example, `/home/dbuser/dblogs/NODE0000/LOGSTREAM0000/`.

If you set the value of the **mirrorlogpath** configuration parameter, the DB2 database system creates active log files in both the log path and the mirror log path. All log data is written to both paths. The mirror log path has a duplicate set of active log files, such that if there is a disk error or human error that corrupts active log files on one of the paths, the database can still function.

In a DB2 pureScale environment, the first member connecting to or activating the database processes the changes to the value of 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. The database manager also attempts to create 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 it or to connect to it fail with SQL5099N. All members must use the same log path.

If you change the mirror log path, there might be log files in the old mirror log path. These log files might not have been archived, so you might need to archive these log files manually. Also, if you are running replication on the related database, replication might still need the log files from before the log path change. If you configured the database to use log archiving and either the DB2 database system automatically archived all the log files or you archived them manually, the DB2 database system can retrieve the log files to complete the replication process. Otherwise, you can copy the files from the old mirror log path to the new mirror log path.

If the **logpath** or **newlogpath** configuration parameter specifies a raw device as the location where the log files are stored, mirror logging, as indicated by the **mirrorlogpath** configuration parameter, is not allowed. If the **logpath** or **newlogpath** configuration parameter specifies a file path as the location where the log files are stored, mirror logging is allowed, and the **mirrorlogpath** configuration parameter must specify a file path.

The **mirrorlogpath** configuration parameter affects log archiving behavior when you also set the **logarchmeth2** configuration parameter. When you set the values of both the **mirrorlogpath** and **logarchmeth2** configuration parameters, the **logarchmeth2** configuration parameter archives log files from the mirror log path instead of archiving additional copies of the log files in the active log path. You can use this log archiving behaviour to improve resilience during rollforward recovery, because a usable archived log file from the mirror log path might still be available to continue a database recovery operation even if a primary log file became corrupted before archiving.

Recommendation:

- Just like the log files, the mirror log files should be on a physical disk that does not have high I/O.
- It is strongly recommended that the mirror log path and the primary log path be on separate devices.

You can use the database system monitor to track the number of I/Os related to database logging. The following data elements return the amount of I/O activity related to database logging.

log_reads
The number of log pages read.

log_writes
The number of log pages written.

You can use an operating system monitor tool to collect information about other disk I/O activity, and then compare the two types of I/O activity.

mon_act_metrics - Monitoring activity metrics configuration parameter

This parameter controls the collection of activity metrics on the entire database and affects activities submitted by connections associated with any DB2 workload definitions.

Configuration type
Database

Parameter type

- Configurable online

Default [range]
BASE [NONE, BASE, EXTENDED]

Upgrade Note
On databases created before V9.7 and then upgraded to V9.7 or higher, the **mon_act_metrics** parameter is set to NONE by default.

If you set this configuration parameter to BASE, all metrics reported through the following interfaces will be collected for all activities executed on the data server, regardless of the DB2 workload the connection that submitted the activity is associated with:

- MON_GET_PKG_CACHE_STMT_DETAILS
- MON_GET_ACTIVITY_DETAILS
- MON_GET_PKG_CACHE_STMT
- Package cache event monitor
- Activity event monitor

If you set this configuration parameter to EXTENDED, the same metrics are collected as under the BASE setting. In addition, the values reported for the following monitor elements are determined with more granularity:

- **total_section_time**
- **total_section_proc_time**
- **total_routine_user_code_time**
- **total_routine_user_code_proc_time**

- **total_routine_time**

For information about how the EXTENDED setting affects these monitor elements, refer to the detailed monitor element descriptions.

If you set this configuration parameter to NONE, the metrics reported through the previously listed interfaces are collected only for the subset of activities submitted by a connection that is associated with a DB2 workload whose COLLECT ACTIVITY METRICS clause has been set to BASE.

mon_deadlock - Monitoring deadlock configuration parameter

This parameter controls the generation of deadlock events at the database level for the lock event monitor.

Configuration type
Database

Parameter type

- Configurable online

Default [range]
WITHOUT_HIST [NONE, WITHOUT_HIST, HISTORY, HIST_AND_VALUES]

If you set this parameter to NONE, no deadlock events will be generated unless deadlock event collection is enabled on DB2 Workload objects using the COLLECT DEADLOCK DATA clause.

If you set the parameter to WITHOUT_HIST, the data about deadlock events are sent to any active locking event monitor when the deadlock event occurs. The data collected for the deadlock wait event will not include a history of activities executed by the application waiting on the deadlock.

If you set the parameter to HISTORY, the lock wait event will include a history of activities executed by the application waiting on the lock. The history only includes activities executed in the current transaction for the application and will only report the newest activities executed if the maximum sized is reached. The default history size is 250. The history size can be configured using the **DB2_MAX_INACT_STMTS** registry variable.

If you set the parameter value to HIST_AND_VALUES, activity history collected with the lock wait event will include the input data values for those activities that have them. These data values will not include LOB data, Start of change LONG VARCHAR data, LONG VARGRAPHIC data, End of change structured type data, or XML data.

This parameter controls the collection deadlock events at the database level for the lock event monitor. The **mon_deadlock** parameter determines whether or not a deadlock wait event will be collected by the lock event monitor when a deadlock occurs.

The **mon_deadlock** parameter value represents a minimum level of collection that is enabled for all DB2 applications. When individual DB2 workloads specify a higher level of collection than the configuration parameter, the DB2 workload setting is used instead of the configuration parameter value. You should note that system applications do not run in a workload and so the **mon_deadlock** parameter is the only way for system applications to collect deadlock data.

To capture the deadlocks with the lock event monitor, as lock waiters or lock holders may span workloads, enable the deadlock collection at the database level. The level of data collected by a deadlock can be controlled individually at the workload level or can be set at the database level by this parameter.

mon_locktimeout - Monitoring lock timeout configuration parameter

This parameter controls the generation of lock timeout events at the database level for the lock event monitor and affects all DB2 workload definitions.

Configuration type
Database

Parameter type

- Configurable online

Default [minimum level of collection that is enabled for all workloads or service classes on the database]
`NONE [NONE, WITHOUT_HIST, HISTORY, HIST_AND_VALUES ]`

If you set this parameter to `NONE`, no lock timeout events will be generated unless lock timeout event collection is enabled on DB2 Workload objects using the COLLECT LOCK TIMEOUT DATA clause.

If you set the parameter to `WITHOUT_HIST`, the data about lock events are sent to any active locking event monitor when the lock event occurs. The data collected for the lock timeout event will not include a history of activities executed by the application waiting on the lock.

If you set the parameter to `HISTORY`, the lock timeout event will include a history of activities executed by the application waiting on the lock. The history only includes activities executed in the current transaction for the application and will only report the newest activities executed if the maximum sized is reached. The default history size is 250. The history size can be configured using the **DB2_MAX_INACT_STMTS** registry variable.

If you set the parameter value to `HIST_AND_VALUES`, activity history collected with the lock timeout event will include the input data values for those activities that have them. These data values will not include LOB data, Start of change LONG VARCHAR data, LONG VARGRAPHIC data, End of change structured type data, or XML data.

This parameter controls the collection of lock timeout events at the database level for the lock event monitor. The **mon_locktimeout** parameter determines whether or not a lock timeout event will be collected by the lock event monitor when an application times out waiting on a lock.

The **mon_locktimeout** parameter value represents a minimum level of collection that is enabled for all DB2 applications. The collection of lock timeout events can be enabled for a subset of DB2 applications using the COLLECT LOCK TIMEOUT DATA clause on a DB2 Workload object instead of the **mon_locktimeout** parameter. When individual DB2 workloads specify a higher level of collection than the configuration parameter, the DB2 workload setting is used instead of the configuration parameter value. You should note that system applications do not run in a workload and so the **mon_locktimeout** parameter is the only way for system applications to collect lock timeout data.

mon_lockwait - Monitoring lock wait configuration parameter

This parameter controls the generation of lock wait events at the database level for the lock event monitor.

Configuration type
Database

Parameter type

- Configurable online

Default [range]
NONE [NONE, WITHOUT_HIST, HISTORY, HIST_AND_VALUES]

If you set this parameter to NONE, no lock wait events will be generated unless lock wait event collection is enabled on DB2 Workload objects using the COLLECT LOCK WAIT DATA clause.

If you set the parameter to WITHOUT_HIST, the data about lock events are sent to any active locking event monitor when the lock event occurs. The data collected for the lock wait event will not include a history of activities executed by the application waiting on the lock.

If you set the parameter to HISTORY, the lock wait event will include a history of activities executed by the application waiting on the lock. The history only includes activities executed in the current transaction for the application and will only report the newest activities executed if the maximum sized is reached. The default history size is 250. The history size can be configured using the **DB2_MAX_INACT_STMTS** registry variable.

If you set the parameter value to HIST_AND_VALUES, activity history collected with the lock wait event will include the input data values for those activities that have them. These data values will not include LOB data, Start of change LONG VARCHAR data, LONG VARGRAPHIC data, End of change structured type data, or XML data.

The **mon_lockwait** parameter value represents a minimum level of collection that is enabled for all DB2 applications. The collection of lock wait events can be enabled for a subset of DB2 applications using the COLLECT LOCK WAIT DATA clause on a DB2 Workload object instead of the **mon_lockwait** parameter. When individual DB2 workloads specify a higher level of collection than the configuration parameter, the DB2 workload setting is used instead of the configuration parameter value.You should note that system applications do not run in a workload and so the **mon_lockwait** parameter is the only way for system applications to collect lock wait data.

This parameter controls the collection of lock wait events at the database level for the lock event monitor. The **mon_lockwait** configuration parameter is used in conjunction with the **mon_lw_thresh** configuration parameter. The **mon_lockwait** parameter determines whether or not a lock wait event will be collected by the lock event monitor when an application waits longer than **mon_lw_thresh** microseconds for a lock.

mon_lw_thresh - Monitoring lock wait threshold configuration parameter

This parameter controls the amount of time spent in lock wait before an event for **mon_lockwait** is generated.

Configuration type
Database

Parameter type

- Configurable online

Default [range]
5000000 [1000 ... MAX_INT]

Upgrade Note
On databases created before V9.7 and then upgraded to V9.7 or higher, the **mon_lw_thresh** parameter is set to 4294967295 by default.

Unit of measure
Microseconds

When this parameter is set both at the database level and at the workload level, the shorter of the two configured times is considered for the given workload.

mon_lck_msg_lvl - Monitoring lock event notification messages configuration parameter

This parameter controls the logging of messages to the administration notification log when lock timeout, deadlock, and lock escalation events occur.

Configuration type
Database

Parameter type
Configurable online

Default [range]
1 [0 - 3]

With the occurrence of lock timeout, deadlock, and lock escalation events, messages can be logged to the administration notification log by setting this database configuration parameter to a value appropriate for the level of notification that you want. The following list outlines the levels of notification that can be set:

0 Level 0: No notification of lock escalations, deadlocks, and lock timeouts is provided

1 Level 1: Notification of lock escalations

2 Level 2: Notification of lock escalations and deadlocks

3 Level 3: Notification of lock escalations, deadlocks, and lock timeouts

The default level of notification setting for this database configuration parameter is 1.

mon_obj_metrics - Monitoring object metrics configuration parameter

This parameter controls the collection of data object metrics on an entire database.

Configuration type
Database

Parameter type

- Configurable online

Default [range]
BASE [NONE, BASE, EXTENDED]

Upgrade Note
On databases created before V9.7 and then upgraded to V9.7 or higher, the **mon_obj_metrics** parameter is set to NONE by default.

Set this configuration parameter to BASE to collect all metrics that are reported through the following table functions:

- MON_GET_BUFFERPOOL
- MON_GET_CONTAINER
- MON_GET_TABLESPACE

Set this configuration parameter to EXTENDED to collect additional metrics that are reported through the following table functions:

Table 137. Metrics returned with EXTENDED option

Table Function	Metrics Reported
MON_GET_INDEX	object_index_gbp_indep_pages_found_in_lbp object_index_gbp_invalid_pages object_index_gbp_l_reads object_index_gbp_p_reads object_index_l_reads object_index_lbp_pages_found object_index_p_reads
MON_GET_INDEX_USAGE_LIST	object_index_gbp_indep_pages_found_in_lbp object_index_gbp_invalid_pages object_index_gbp_l_reads object_index_gbp_p_reads object_index_l_reads object_index_lbp_pages_found object_index_p_reads
MON_GET_TABLE	direct_read_reqs direct_reads direct_write_reqs direct_writes lock_escals lock_escals_global lock_wait_time lock_wait_time_global lock_waits lock_waits_global object_data_gbp_indep_pages_found_in_lbp object_data_gbp_invalid_pages object_data_gbp_l_reads object_data_gbp_p_reads object_data_l_reads object_data_lbp_pages_found object_data_p_reads object_xda_gbp_indep_pages_found_in_lbp object_xda_gbp_invalid_pages object_xda_gbp_l_reads object_xda_gbp_p_reads object_xda_l_reads object_xda_lbp_pages_found object_xda_p_reads

Table 137. Metrics returned with EXTENDED option (continued)

Table Function	Metrics Reported
MON_GET_TABLE_USAGE_LIST	direct_read_reqs direct_reads direct_write_reqs direct_writes lock_escals lock_escals_global lock_wait_time lock_wait_time_global lock_waits lock_waits_global object_data_gbp_indep_pages_found_in_lbp object_data_gbp_invalid_pages object_data_gbp_l_reads object_data_gbp_p_reads object_data_l_reads object_data_lbp_pages_found object_data_p_reads object_xda_gbp_indep_pages_found_in_lbp object_xda_gbp_invalid_pages object_xda_gbp_l_reads object_xda_gbp_p_reads object_xda_l_reads object_xda_lbp_pages_found object_xda_p_reads overflow_accesses overflow_creates rows_deleted rows_inserted rows_read rows_updated

If you set this configuration parameter to NONE, the metrics reported through the previously mentioned table functions are not collected.

mon_pkglist_sz - Monitoring package list size configuration parameter

This parameter controls the maximum number of entries that can appear in the package listing per unit of work as captured by the unit of work event monitor.

Configuration type
Database

Parameter type
Configurable online

Propagation clause
Next unit of work

Default [range]
32 [0 - 1024]

Unit of measure
Number of entries in the package list

The package list will have a maximum size as specified by the value for this database configuration parameter. The size of the package list is determined at the

start of the unit of work. Changes to the package list size are not reflected until the following unit of work. The default size for the package list is 32 entries.

mon_req_metrics - Monitoring request metrics configuration parameter

This parameter controls the collection of request metrics on the entire database and affects requests executing in any DB2 service classes.

Configuration type
Database

Parameter type

- Configurable online

Default [range]
BASE [NONE, BASE, EXTENDED]

Upgrade Note
On databases created before V9.7 and then upgraded to V9.7 or higher, the **mon_req_metrics** parameter is set to NONE by default.

If you set this configuration parameter to BASE, all metrics reported through the following interfaces are collected for all requests executed on the data server, irrespective of the DB2 service class the request runs in:

- MON_GET_UNIT_OF_WORK
- MON_GET_UNIT_OF_WORK_DETAILS
- MON_GET_CONNECTION
- MON_GET_CONNECTION_DETAILS
- MON_GET_SERVICE_SUBCLASS
- MON_GET_SERVICE_SUBCLASS_DETAILS
- MON_GET_WORKLOAD
- MON_GET_WORKLOAD_DETAILS
- Statistics event monitor (DETAILS_XML monitor element in the event_wlstats and event_scstats logical data groups)
- Unit of work event monitor

If you set this configuration parameter to EXTENDED, the same metrics are collected as under the BASE setting. In addition, the values reported for the following monitor elements are determined with more granularity:

- **total_section_time**
- **total_section_proc_time**
- **total_routine_user_code_time**
- **total_routine_user_code_proc_time**
- **total_routine_time**

For information about how the EXTENDED setting affects these monitor elements, refer to the detailed monitor element descriptions.

If you set this configuration parameter to NONE, the metrics reported through the previously listed interfaces, are collected only for the subset of requests running in a DB2 service class whose service superclass has the COLLECT REQUEST METRICS clause set to BASE.

mon_uow_data - Monitoring unit of work events configuration parameter

This parameter controls the generation of unit of work events at the database level for the unit of work event monitor and affects units of work on the data server. It is a parent parameter to the **mon_uow_execlist** and **mon_uow_pkglist** configuration parameters.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Propagation class
Next unit of work

Default [range]
NONE [NONE, BASE]

Valid values for this parameter are as follows:

NONE All data collection for a UOW event monitor is disabled.

BASE Basic data collection for a UOW event monitor is enabled.

This parameter specifies whether information about a unit of work, also referred to as a transaction, should be sent to the active unit of work event monitors when the unit of work is completed.

When this parameter is disabled, all of its child parameters are also disabled, but their settings, as recorded in the database configuration file, do not change. When this parent parameter is enabled, recorded values for its child parameters take effect. Thus, data collection can be enabled or disabled globally.

If the parameter is set to BASE, information about all units of work that are executed on the data server are sent to the active unit of work event monitors when the units of work are completed. If the parameter is set to NONE, information is sent to the unit of work event monitors only for those units of work that are executed under a DB2 workload whose COLLECT UNIT OF WORK DATA clause is set to BASE. The default setting is NONE.

The information that is gathered at the end of a unit of work includes the system-level request metrics for that unit of work, for example, the amount of CPU that is used during the unit of work. The collection of these request metrics is controlled independently from the collection of the unit of work data. You collect unit of work data by using either the COLLECT REQUEST METRICS clause on a DB2 service superclass or setting the **mon_req_metrics** database configuration parameter to the value of BASE. If you do not enable request metrics collection, the value of all the request metrics that are gathered as part of the unit of work data is 0.

mon_uow_execlist - Monitoring unit of work events with executable list configuration parameter

This parameter controls the generation of unit of work events with executable ID listing information included. This is done at the database level for the unit of work event monitor. The **mon_uow_execlist** database configuration parameter is a child parameter of the **mon_uow_data** database configuration parameter.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Propagation class
Next unit of work

Default [range]
`OFF [OFF, ON]`

Valid values for this parameter are as follows:

OFF Executable ID list collection is disabled for a unit of work event monitor.

ON Executable ID list collection is enabled for a unit of work event monitor.

mon_uow_pkglist - Monitoring unit of work events with package list configuration parameter

This parameter controls the generation of unit of work events with package listing information included.This is done at the database level for the unit of work event monitor. The **mon_uow_pkglist** database configuration parameter is a child parameter of the **mon_uow_data** database configuration parameter.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Propagation class
Next unit of work

Default [range]
`OFF [OFF, ON]`

Valid values for this parameter are as follows:

OFF Package list collection is disabled for a unit of work event monitor.

ON Package list collection is enabled for a unit of work event monitor.

multipage_alloc - Multipage file allocation enabled

Multipage file allocation is used to improve insert performance. It applies to SMS table spaces only. If enabled, all SMS table spaces are affected: there is no selection possible for individual SMS table spaces.

Configuration type
Database

Parameter type
Informational

The default for the parameter is Yes: multipage file allocation is enabled.

Following database creation, this parameter cannot be set to No. Multipage file allocation cannot be disabled once it has been enabled. The **db2empfa** tool can be used to enable multipage file allocation for a database that currently has it disabled.

newlogpath - Change the database log path

This parameter allows you to specify a string of up to 242 bytes to change the location where the log files are stored.

Configuration type
Database

Parameter type
Configurable

Default [range]
Null [any valid path or device]

The string can point to either a path name or to a raw device.

Important: The use of raw devices for database logging has been deprecated starting in Version 9.1 and is not supported in DB2 pureScale environments. Where supported, the database manager uses automatically non-buffered I/O for the active log files and all other log file access will be buffered.

If the string points to a path name, it must be a fully qualified path name, not a relative path name.

In both DB2 pureScale environments and DB2 Enterprise Server Edition environments, the database partition number and a log stream ID are automatically appended to the path, for example, `/home/dbuser/dblogs/NODE0000/LOGSTREAM0000/`.

In a DB2 pureScale environment, the first member connecting to or activating the database processes configuration 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 specified path is accepted, the new value is propagated to each member. If a member fails while trying to switch to the new path, subsequent attempts to activate it or to connect to it will fail (SQL5099N). All members must use the same log path.

If you want to use replication, and your log path is a raw device, the **overflowlogpath** configuration parameter must be configured.

To specify a device, specify a string that the operating system identifies as a device. For example:

- On Windows, `\\.\d:` or `\\.\PhysicalDisk5`

  Note: You must have Windows Version 4.0 with Service Pack 3 or later installed to be able to write logs to a device.
- On Linux and UNIX platforms, `/dev/rdblog8`

Note: You can only specify a device on AIX, Windows 2000, Windows, Solaris, HP-UX, and Linux platforms.

The new setting does not become the value of **logpath** until both of the following events occur:

- The database is in a consistent state, as indicated by the **database_consistent** parameter.
- All applications are disconnected from the database

When the first new connection is made to the database, the database manager will move the logs to the new location specified by **logpath**.

There might be log files in the old log path. These log files might not have been archived. You might need to archive these log files manually. Also, if you are running replication on this database, replication might still need the log files from before the log path change. If the database is configured to use log archiving and if all the log files have been archived either by the DB2 database system automatically or by yourself manually, then the DB2 database system will be able to retrieve the log files to complete the replication process. Otherwise, you can copy the files from the old log path to the new log path.

If **logpath** or **newlogpath** specifies a raw device as the location where the log files are stored, mirror logging, as indicated by **mirrorlogpath**, is not allowed. If **logpath** or **newlogpath** specifies a file path as the location where the log files are stored, mirror logging is allowed and **mirrorlogpath** must also specify a file path.

Recommendation: Ideally, the log files will be on a physical disk which does **not** have high I/O. For instance, avoid putting the logs on the same disk as the operating system or high volume databases. This will allow for efficient logging activity and will reduce additional processing time taken, such as when waiting for I/O.

You can use the database system monitor to track the number of I/Os related to database logging.

The monitor elements **log_reads** (number of log pages read) and **log_writes** (number of log pages written) return the amount of I/O activity related to database logging. You can use an operating system monitor tool to collect information about other disk I/O activity, then compare the two types of I/O activity.

Do not use a network or local file system that is shared as the log path for both the primary and standby databases in a DB2 High Availability Disaster Recovery (HADR) database pair. The primary and standby databases each have copies of the

transaction logs - the primary database ships logs to the standby database. If the log path for both the primary and standby databases points to the same physical location, then the primary and standby database would use the same physical files for their corresponding copies of the logs. The database manager returns an error if the database manager detects a shared log path.

num_db_backups - Number of database backups

This parameter specifies the number of database backups to retain for a database.

Configuration type
Database

Parameter type
Configurable online

Propagation class
Transaction boundary

Default [range]
12 [1 - 32 767]

After the specified number of backups is reached, old backups are marked as expired in the recovery history file. Recovery history file entries for the table space backups and load copy backups that are related to the expired database backup are also marked as expired. When a backup is marked as expired, the physical backups can be removed from where they are stored (for example, disk, tape, TSM). The next database backup will prune the expired entries from the recovery history file.

The **rec_his_retentn** configuration parameter should be set to a value compatible with the value of **num_db_backups**. For example, if **num_db_backups** is set to a large value, the value for **rec_his_retentn** should be large enough to support the number of backups set as **num_db_backups**.

num_freqvalues - Number of frequent values retained

This parameter allows you to specify the number of "most frequent values" that will be collected when the **WITH DISTRIBUTION** option is specified on the **RUNSTATS** command.

Configuration type
Database

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
10 [0 - 32 767]

Unit of measure
Counter

Increasing the value of this parameter increases the amount of statistics heap (**stat_heap_sz**) used when collecting statistics.

The "most frequent value" statistics help the optimizer understand the distribution of data values within a column. A higher value results in more information being

available to the query optimizer but requires additional catalog space. When `0` is specified, no frequent-value statistics are retained, even if you request that distribution statistics be collected.

You can also specify the number of frequent values retained as part of the **RUNSTATS** command at the table or the column level by using the **NUM_FREQVALUES** command parameter. If none is specified, the **num_freqvalues** configuration parameter value is used. Changing the number of frequent values retained through the **RUNSTATS** command is easier than making the change using the **num_freqvalues** database configuration parameter.

Updating this parameter can help the optimizer obtain better selectivity estimates for some predicates (=, <, >) over data that is non-uniformly distributed. More accurate selectivity calculations might result in the choice of more efficient access plans.

After changing the value of this parameter, you need to:

- Run the **RUNSTATS** command again to collect statistics with the changed number of frequent values
- Rebind any packages containing static SQL or XQuery statements.

When using **RUNSTATS**, you have the ability to limit the number of frequent values collected at both the table level and the column level. This allows you to optimize on space occupied in the catalogs by reducing the distribution statistics for columns where they could not be exploited and yet still using the information for critical columns.

Recommendation: In order to update this parameter you should determine the degree of non-uniformity in the most important columns (in the most important tables) that typically have selection predicates. This can be done using an SQL SELECT statement that provides an ordered ranking of the number of occurrences of each value in a column. You should not consider uniformly distributed, unique, long, or LOB columns. A reasonable practical value for this parameter lies in the range of `10` to `100`.

Note that the process of collecting frequent value statistics requires significant CPU and memory (**stat_heap_sz**) resources.

num_iocleaners - Number of asynchronous page cleaners

This parameter allows you to specify the number of asynchronous page cleaners for a database.

Configuration type
Database

Parameter type

- Configurable
- Configurable by member in a DB2 pureScale environment

Default [range]
`Automatic [0 - 255 ]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Counter

The number of castout engines is equal to the value that is set for the **num_iocleaners** configuration parameter. However, the maximum number of castout engines is 128.

These page cleaners write changed pages from the buffer pool to disk before the space in the buffer pool is required by a database agent. As a result, database agents should not have to wait for changed pages to be written out so that they might use the space in the buffer pool. This improves overall performance of the database applications.

The page cleaners will also decrease recovery time from soft failures, such as power outages, because the contents of the database on disk will be more up-to-date at any given time.

If this parameter is set to AUTOMATIC, the number of page cleaners started will be based on the number of physical CPU cores configured on the current machine, as well as the number of local logical database partitions in a partitioned database environment. There will always be at least one page cleaner started when this parameter is set to AUTOMATIC.

The number of page cleaners to start when this parameter is set to AUTOMATIC will be calculated using the following formula:

```
number of page cleaners = max( ceil( # CPUs / # local logical DPs ) - 1, 1 )
```

This formula ensures that the number of page cleaners is distributed almost evenly across your logical database partitions, and that there are no more page cleaners than there are physical CPU cores.

Note: On the HP-UX platform, we will use the number of logical CPUs in the calculation, instead of the number of physical CPU cores.

Recommendation: Consider the following factors when setting the value for this parameter:

- Workload

  Environments with high update transaction rates might require more page cleaners to be configured. This is only applicable when **DB2_USE_ALTERNATE_PAGE_CLEANING** is OFF, which is also the default value.
- Buffer pool sizes

  Environments with large buffer pools might also require more page cleaners to be configured. This is only applicable when **DB2_USE_ALTERNATE_PAGE_CLEANING** is OFF, which is also the default value.

You can use the database system monitor to help you tune this configuration parameter using information from the event monitor about write activity from a buffer pool:

- The parameter can be reduced if both of the following conditions are true:
  - **pool_data_writes** is approximately equal to **pool_async_data_writes**
  - **pool_index_writes** is approximately equal to **pool_async_index_writes**.

    Note: Reducing to a lower value, than what is computed via AUTOMATIC, is not recommended.

- The parameter should be increased if either of the following conditions are true:
  - **pool_data_writes** is much greater than **pool_async_data_writes**
  - **pool_index_writes** is much greater than **pool_async_index_writes**.

num_ioservers - Number of I/O servers

This parameter specifies the number of I/O servers for a database. No more than this number of I/Os for prefetching and utilities can be in progress for a database at any time.

Configuration type
Database

Parameter type
- Configurable
- Configurable by member in a DB2 pureScale environment

Default [range]
`Automatic [1 - 255 ]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Counter

When allocated
When an application connects to a database

When freed
When an application disconnects from a database

I/O servers, also called prefetchers, are used on behalf of the database agents to perform prefetch I/O and asynchronous I/O by utilities such as backup and restore. An I/O server waits while an I/O operation that it initiated is in progress. Non-prefetch I/Os are scheduled directly from the database agents and as a result are not constrained by **num_ioservers**.

If this parameter is set to `AUTOMATIC`, the number of prefetchers started will be based on the parallelism settings of the table spaces in the current database partition.

When this parameter is set to `AUTOMATIC`, the number of prefetchers to start will be calculated at database activation time based on the following formula:

```
number of prefetchers = max( max over all table spaces( parallelism setting ), 3 )
```

Where parallelism settings are controlled by the **DB2_PARALLEL_IO** system environment variable.

Recommendation: In order to fully exploit all the I/O devices in the system, a good value to use is generally one or two more than the number of physical devices on which the database resides. It is better to configure additional I/O servers, since there is some additional processing time associated with each I/O server and any unused I/O servers will remain idle.

num_log_span - Number log span

This parameter specifies whether there is a limit to how many log files one transaction can span, and what that limit is.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
0 [0 - 65 535]

Unit of measure
Counter

If the value is not 0, this parameter indicates the number of active log files that one active transaction is allowed to span.

If the value is set to 0, there is no limit to how many log files one single transaction can span.

If an application violates the **num_log_span** configuration, the application is forced to disconnect from the database and the transaction is rolled back.

This parameter, along with the **max_log** configuration parameter, can be useful when enabling infinite active logspace. If infinite logging is on (that is, if *logsecond* is set to -1) then transactions are not restricted to the upper limit of the number of log files (*logprimary* + *logsecond*). When the value of *logprimary* is reached, DB2 starts to archive the active logs, rather than failing the transaction. This can cause problems if, for example, an application contains a long running transaction that is left uncommitted. If this occurs, the active logspace continues to grow, possibly leading to poor crash recovery performance. To prevent this, you can specify values for one or both of the **max_log** or **num_log_span** configuration parameters.

Note: The following DB2 commands are excluded from the limitation imposed by the **num_log_span** configuration parameter: ARCHIVE LOG, BACKUP DATABASE, LOAD, REORG, RESTORE DATABASE, and ROLLFORWARD DATABASE.

num_quantiles - Number of quantiles for columns

This parameter controls the number of quantiles that will be collected when the **WITH DISTRIBUTION** option is specified on the **RUNSTATS** command.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
20 [0 - 32 767]

Unit of measure
Counter

Increasing the value of this parameter increases the amount of statistics heap (**stat_heap_sz**) used when collecting statistics.

The "quantile" statistics help the optimizer understand the distribution of data values within a column. A higher value results in more information being available to the query optimizer but requires additional catalog space. When `0` or `1` is specified, no quantile statistics are retained, even if you request that distribution statistics be collected.

You can also specify the number of quantiles collected as part of the **RUNSTATS** command at the table or the column level, by using the **NUM_QUANTILES** command parameter. If none is specified, the **num_quantiles** configuration parameter value is used. Changing the number of quantiles that will be collected through the **RUNSTATS** command is easier than making the change using the **num_quantiles** database configuration parameter.

Updating this parameter can help obtain better selectivity estimates for range predicates over data that is non-uniformly distributed. Among other optimizer decisions, this information has a strong influence on whether an index scan or a table scan will be chosen. (It is more efficient to use a table scan to access a range of values that occur frequently and it is more efficient to use an index scan for a range of values that occur infrequently.)

After changing the value of this parameter, you need to:

- Run the **RUNSTATS** command again to collect statistics with the changed number of frequent values
- Rebind any packages containing static SQL or XQuery statements.

When using **RUNSTATS**, you have the ability to limit the number of quantiles collected at both the table level and the column level. This allows you to optimize on space occupied in the catalogs by reducing the distribution statistics for columns where they could not be exploited and yet still using the information for critical columns.

Recommendation: The default value for this parameter provides reasonably accurate estimates in most cases. You can consider increasing the value if you observe significant and consistent differences between:

- Selectivity estimates from the explain output; and
- Actual selectivity of range predicates over non-uniformly distributed column data.

A reasonable practical value for this parameter lies in the range of 10 to 50.

numarchretry - Number of retries on error

This parameter specifies the number of times that the DB2 database system is to try archiving a log file to the primary or the secondary archive directory before trying to archive log files to the failover directory.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter type

- Configurable online

Default [range]

5 [0 - 65 535]

This parameter is only used if the **failarchpath** database configuration parameter is set. If **numarchretry** is not set, the DB2 database system will continuously retry archiving to the primary or the secondary log path.

numsegs - Default number of SMS containers

This parameter has been deprecated since Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the database manager in DB2 Version 9.5 or later releases.

Note: The following information applies only to pre-Version 9.5 data servers and clients.

Configuration type

Database

Parameter type

Informational

Unit of measure

Counter

This parameter indicates the number of containers that will be created within the default table spaces. It also shows the information used when you created your database, whether it was specified explicitly or implicitly on the **CREATE DATABASE** command.

This parameter only applies to SMS table spaces; the CREATE TABLESPACE statement **does not** use it in any way.

number_compat - Number compatibility database configuration parameter

This parameter indicates whether the compatibility semantics associated with the NUMBER data type are applied to the connected database.

Configuration type

Database

Parameter type

Informational

The value is determined at database creation time, and is based on the setting of the **DB2_COMPATIBILITY_VECTOR** registry variable for NUMBER support. The value cannot be changed.

overflowlogpath - Overflow log path

The **overflowlogpath** parameter specifies a location for DB2 databases to find log files needed for a rollforward operation, as well as where to store active log files retrieved from the archive. It also gives a location for finding and storing log files needed for using db2ReadLog API.

Configuration type
Database

Parameter type
Configurable online

Propagation class
Immediate

Default [range]
NULL [any valid path]

This parameter can be used for several functions, depending on your logging requirements.

- Use the **overflowlogpath** parameter to specify a location for DB2 databases to find log files that are needed for a rollforward operation. It is similar to the **OVERFLOW LOG PATH** option on the **ROLLFORWARD** command. Instead of always specifying **OVERFLOW LOG PATH** on every **ROLLFORWARD** command, you can set this configuration parameter once. However, if both are used, the **OVERFLOW LOG PATH** option overwrites the **overflowlogpath** configuration parameter, for that particular rollforward operation.
- If **logsecond** is set to **-**1, you can specify a directory for DB2 to store active log files retrieved from the archive with the **overflowlogpath** parameter. Active log files have to be retrieved for rollback operations if they are no longer in the active log path. Without the **overflowlogpath** parameter set, DB2 databases retrieve the log files into the active log path. Use the **overflowlogpath** parameter to provide additional resource for DB2 databases to store the retrieved log files. The benefit includes spreading the I/O cost to different disks, and allowing more log files to be stored in the active log path.
- If you need to use the db2ReadLog API for replication, for example, use the **overflowlogpath** to specify a location for DB2 databases to search for log files that are needed for this API. Before DB2 Version 8, db2ReadLog was called sqlurlog. If the log file is not found (in either the active log path or the overflow log path) and the database is configured to archive logs by using the **logarchmeth1** or **logarchmeth2** parameters, DB2 retrieves the log file. Also use the **overflowlogpath** parameter to specify a directory for DB2 databases to store the log files retrieved. The benefit comes from reducing the I/O cost on the active log path and allowing more log files to be stored in the active log path.
- If you have configured a raw device for the active log path, the **overflowlogpath** parameter must be configured if you want to set the **logsecond** parameter to **-**1, or if you want to use the db2ReadLog API.
- You can specify a location for DB2 databases to retrieve log files that are required for a **BACKUP DATABASE INCLUDE LOGS** operation.

To set **overflowlogpath**, specify a string of up to 242 bytes. The string must point to a path name, and it must be a fully qualified path name, not a relative path name. The path name must be a directory, not a raw device.

Note: In both DB2 pureScale environments and DB2 Enterprise Server Edition environments, the database partition number and a log stream ID are automatically appended to the path, for example, `/home/dbuser/dblogs/NODE0000/LOGSTREAM0000/`.

pagesize - Database default page size

This parameter contains the value that was used as the default page size when the database was created. Possible values are: 4 096, 8 192, 16 384 and 32 768. When a buffer pool or table space is created in that database, the same default page size applies.

Configuration type
Database

Parameter type
Informational

pckcachesz - Package cache size

This parameter is allocated out of the database shared memory, and is used for caching of sections for static and dynamic SQL and XQuery statements on a database.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]

32-bit operating systems
`Automatic [-1, 32 - 128 000]`

64-bit operating systems
`Automatic [-1, 32 - 2 147 483 646]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
When the database is initialized

When freed
When the database is shut down

In a partitioned database system, there is one package cache for each database partition.

Caching packages allows the database manager to reduce its internal processing time by eliminating the need to access the system catalogs when reloading a package; or, in the case of dynamic SQL or XQuery statements, eliminating the need for compilation. Sections are kept in the package cache until one of the following events occurs:

- The database is shut down
- The package or dynamic SQL or XQuery statement is invalidated
- The cache runs out of space.

This caching of the section for a static or dynamic SQL or XQuery statement can improve performance, especially when the same statement is used multiple times by applications connected to a database. This is particularly important in a transaction processing environment.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. When **self_tuning_mem** is set to ON, the memory tuner will dynamically size the memory area controlled by **pckcachesz** as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the **self_tuning_mem** configuration parameter is set to ON.)

When this parameter is set to -1, the value used to calculate the page allocation is eight times the value specified for the **maxappls** configuration parameter. The exception to this occurs if eight times **maxappls** is less than 32. In this situation, the default value of -1 will set **pckcachesz** to 32.

Recommendation: When tuning this parameter, you should consider whether the extra memory being reserved for the package cache might be more effective if it was allocated for another purpose, such as the buffer pool or catalog cache. For this reason, you should use benchmarking techniques when tuning this parameter.

Tuning this parameter is particularly important when several sections are used initially and then only a few are run repeatedly. If the cache is too large, memory is wasted holding copies of the initial sections.

The following monitor elements can help you determine whether you should adjust this configuration parameter:

- **pkg_cache_lookups** (package cache lookups)
- **pkg_cache_inserts** (package cache inserts)
- **pkg_cache_size_top** (package cache high water mark)
- **pkg_cache_num_overflows** (package cache overflows)

Note: The package cache is a working cache, so you cannot set this parameter to zero. There must be sufficient memory allocated in this cache to hold all sections of the SQL or XQuery statements currently being executed. If there is more space allocated than currently needed, then sections are cached. These sections can simply be executed the next time they are needed without having to load or compile them.

The limit specified by the **pckcachesz** parameter is a soft limit. This limit can be exceeded, if required, if memory is still available in the database shared set. You can use the **pkg_cache_size_top** monitor element to determine the largest that the package cache has grown, and the **pkg_cache_num_overflows** monitor element to determine how many times the limit specified by the **pckcachesz** parameter has been exceeded.

priv_mem_thresh - Private memory threshold

This parameter has been deprecated since Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the database manager in DB2 Version 9.5 or later releases.

Note: The following information applies only to pre-Version 9.5 data servers and clients.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
`20 000 [-1; 32 - 112 000]`

Unit of measure
Pages (4 KB)

This parameter is used to determine the amount of unused agent private memory that will be kept allocated, ready to be used by new agents that are started. It does not apply to Linux and UNIX platforms.

A value of `-1` will cause this parameter to use the value of the **`min_priv_mem`** parameter.

Recommendation: When setting this parameter, you should consider the client connection/disconnection patterns as well as the memory requirements of other processes on the same machine.

If there is only a brief period during which many clients are concurrently connected to the database, a high threshold will prevent unused memory from being decommitted and made available to other processes. This case results in poor memory management which can affect other processes which require memory.

If the number of concurrent clients is more uniform and there are frequent fluctuations in this number, a high threshold will help to ensure memory is available for the client processes and can reduce the processing time taken to allocate and deallocate memory.

rec_his_retentn - Recovery history retention period

This parameter specifies the number of days that historical information about the backups will be retained.

Configuration type
Database

Parameter type
Configurable

Default [range]
366 [-1; 0 - 30 000]

Unit of measure
Days

If the recovery history file is not needed to keep track of backups, restores, and loads, this parameter can be set to a small number.

If **rec_his_retentn** is set to -1 and **auto_del_rec_obj** is set to OFF, the number of entries indicating full database backups (and any table space backups that are associated with the database backup) will correspond with the value specified by the **num_db_backups** database configuration parameter. Other entries in the recovery history file can only be pruned by explicitly using the available commands or APIs. If **rec_his_retentn** is set to -1 and **auto_del_rec_obj** is set to ON, the history file is not automatically pruned and no recovery objects are deleted.

If **rec_his_retentn** is set to 0 and **auto_del_rec_obj** is set to OFF, all entries in the history file, except the last full backup, are pruned. If **auto_del_rec_obj** is set to ON, automated history file pruning and recovery object deletion are carried out based on the timestamp of the backup selected by the **num_db_backups** database configuration parameter.

No matter how small the retention period, the most recent full database backup plus its restore set will always be kept, unless you use the **PRUNE HISTORY** command with the **FORCE** option.

restore_pending - Restore pending

This parameter states whether a RESTORE PENDING status exists in the database.

Configuration type
Database

Parameter type
Informational

restrict_access - Database has restricted access configuration parameter

This parameter indicates whether the database was created using the restrictive set of default actions. In other words, if it was created with the **RESTRICTIVE** clause in the **CREATE DATABASE** command.

Configuration type
Database

Parameter type
Informational

YES The **RESTRICTIVE** clause was used in the **CREATE DATABASE** command when this database was created.

NO The **RESTRICTIVE** clause was not used in the **CREATE DATABASE** command when this database was created.

rollfwd_pending - Roll forward pending indicator

This parameter informs you whether or not a rollforward recovery is required, and where it is required.

Configuration type
Database

Parameter type
Informational

This parameter can indicate one of the following states:

- DATABASE, meaning that a rollforward recovery procedure is required for this database
- TABLESPACE, meaning that one or more table spaces need to be rolled forward
- NO, meaning that the database is usable and no rollforward recovery is required.

The recovery (using **ROLLFORWARD DATABASE**) must complete before you can access the database or table space.

section_actuals - Section actuals configuration parameter

This parameter enables collection of section actuals (runtime statistics that are measured during section execution) such that the statistics can be viewed when an event monitor is subsequently created.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Propagation class
Unit of work boundary

Default [range]
NONE [NONE, BASE]

The following values are valid for this parameter:

NONE Specifies that collection of section actuals is disabled.

BASE Specifies that measurement is enabled and the following statistics are collected:

- Basic operator cardinality counts
- Statistics for each object that is referenced during section execution (DML statements only)

After section actuals are enabled, capture section actuals by using an activity event monitor and view them through a section explain that you perform by using the EXPLAIN_FROM_ACTIVITY procedure.

You cannot enable this parameter if the **auto_stats_prof** database configuration parameter is enabled (SQLCODE -5153).

self_tuning_mem- Self-tuning memory

This parameter determines whether the memory tuner will dynamically distribute available memory resources as required between memory consumers that are enabled for self-tuning.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]

Single-database partition environments
ON [ON; OFF]

Multi-database partition environments
OFF [ON; OFF]

In a database that is upgraded from an earlier version, **self_tuning_mem** will retain the old value.

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Because memory is being traded between memory consumers, there must be at least two memory consumers enabled for self-tuning in order for the memory tuner to be active. When **self_tuning_mem** is set to ON, but there are less than two memory consumers enabled for self-tuning, the memory tuner is inactive. (The exception to this is the sort heap memory area, which can be tuned regardless of whether other memory consumers are enabled for self-tuning or not.)

This parameter is ON by default in single database partition environments. In multi-database partition environments, it is OFF by default.

The memory consumers that can be enabled for self-tuning include:

- Package cache (controlled by the **pckcachesz** configuration parameter)
- Lock List (controlled by the **locklist** and **maxlocks** configuration parameters)
- Sort heap (controlled by the **sheapthres_shr** and **sortheap** configuration parameters)
- Database shared memory (controlled by the **database_memory** configuration parameter)
- Buffer pools (controlled by the size parameter of the ALTER BUFFERPOOL and CREATE BUFFERPOOL statements)

Note: In DB2 pureScale environments, only the local buffer pools (LBPs) can be managed using the self-tuning memory feature. Memory for group buffer pools (GBPs) is allocated and controlled using the **cf_gbp_sz** configuration parameter.

To view the current setting for this parameter, use the **GET DATABASE CONFIGURATION** command specifying the **SHOW DETAIL** parameter. The possible settings returned for this parameter are:

```
Self Tuning Memory          (SELF_TUNING_MEM) = OFF
Self Tuning Memory          (SELF_TUNING_MEM) = ON (Active)
Self Tuning Memory          (SELF_TUNING_MEM) = ON (Inactive)
Self Tuning Memory          (SELF_TUNING_MEM) = ON
```

The following values indicate:

- ON (Active) - the memory tuner is actively tuning the memory on the system
- ON (Inactive) - that although the parameter is set ON, self-tuning is not occurring because there are less than two memory consumers enabled for self-tuning, or the database or instance is in quiesce mode.
- ON without (Active) or (Inactive) - from a query without the **SHOW DETAIL** option, or without a database connection.

In partitioned environments, the **self_tuning_mem** configuration parameter will only show ON (Active) for the database partition on which the tuner is running. On all other nodes **self_tuning_mem** will show ON (Inactive). As a result, to determine if the memory tuner is active in a partitioned database, you must check the **self_tuning_mem** parameter on all database partitions.

If you have upgraded to DB2 Version 9 from an earlier version of DB2 and you plan to use the self-tuning memory feature, you should configure the following health indicators to disable threshold or state checking:

- Shared Sort Memory Utilization - **db.sort_shrmem_util**
- Percentage of sorts that overflowed - **db.spilled_sorts**
- Long Term Shared Sort Memory Utilization - **db.max_sort_shrmem_util**
- Lock List Utilization - **db.locklist_util**
- Lock Escalation Rate - **db.lock_escal_rate**
- Package Cache Hit Ratio - **db.pkgcache_hitratio**

One of the objectives of the self-tuning memory feature is to avoid having memory allocated to a memory consumer when it is not immediately required. Therefore, utilization of the memory allocated to a memory consumer might approach 100% before more memory is allocated. By disabling these health indicators, you will avoid unnecessary alerts triggered by the high rate of memory utilization by a memory consumer.

Instances created in DB2 Version 9 will have these health indicators disabled by default.

seqdetect - Sequential detection and readahead flag

This parameter controls whether the database manager is allowed to perform sequential detection or readahead prefetching during I/O activity.

Configuration type
Database

Parameter type
Configurable online

Propagation class
Immediate

Default [range]
Yes [Yes; No]

The database manager can monitor I/O during index scans, and if sequential page reading is occurring the database manager can activate I/O prefetching. This type of sequential prefetch is known as *sequential detection*. However, at runtime, the prefetching type might switch from sequential detection prefetching to readahead prefetching when it detects that sequential detection prefetching is not working well enough.

If this parameter is set to No, both sequential detection and readahead prefetching are disabled for the **INSPECT** command with the **INDEXDATA** option, the **RUNSTATS** command, and all index scans during query execution. However, for the **REORG** command with the **CLEANUP** option for indexes, setting **seqdetect** to No disables only sequential detection prefetching for indexes.

Recommendation: In most cases, you should use the default value for this parameter. Try turning **seqdetect** off, only if other tuning efforts were unable to correct serious query performance problems.

sheapthres_shr - Sort heap threshold for shared sorts

This parameter represents a soft limit on the total amount of database shared memory that can be used by sort memory consumers at any one time.

Configuration type
Database

Applies to
OLAP functions

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]

32-bit platforms
`Automatic [250 - 524 288]`

64-bit platforms
`Automatic [250 - 2 147 483 647]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

There are other sort memory consumers in addition to sort, like hash join, index ANDing, block index ANDing, merge join, and in-memory tables. When the total amount of shared memory for shared sort memory consumers approaches the **sheapthres_shr** limit, a memory throttling mechanism is activated and the future shared sort memory consumer requests might be granted less memory than requested, but will always be granted more than the minimum they need for finishing the task. Once the **sheapthres_shr** limit is exceeded, all requests of shared sort memory from sort memory consumers will be granted the minimum amount of memory required to finish the task. When the total amount of shared memory for active shared sort memory consumers reaches this limit, subsequent sorts could fail (SQL0955C).

When the value of the database manager configuration parameter **sheapthres** is `0`, all sort memory consumers for the database will use the database shared memory with **sheapthres_shr** instead of private sort memory.

When **sheapthres_shr** is set to `AUTOMATIC`, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this

parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active. Memory consumers include **sheapthres_shr**, **pckcachesz**, buffer pool (each buffer pool counts as one), **locklist**, and **database_memory**.

Automatic tuning of **sheapthres_shr** is allowed only when the database manager configuration parameter **sheapthres** is set to 0.

The value of **sortheap** is tuned together with the **sheapthres_shr** parameter therefore disabling self tuning of the **sortheap** parameter automatically disables self tuning of the **sheapthres_shr** parameter. Enabling self tuning of the **sheapthres_shr** parameter automatically enables self tuning of the **sortheap** parameter.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the **self_tuning_mem** configuration parameter is set to ON).

When the value of this parameter is updated online, only new requests of shared-sort memory made after the update will use the new value. It is recommended that you reduce the value of **sortheap** before reducing the value of **sheapthres_shr** and to increase the value of **sheapthres_shr** before increasing the value of **sortheap**.

When the database manager configuration parameter **sheapthres** is greater than 0, **sheapthres_shr** is only meaningful in two cases:

- LOADing into an XML table requires shared sort memory. In this case, we require **sheapthres_shr** to be non-zero or else an error will be returned that the shared sort memory could not be allocated for this utility.
- If the **intra_parallel** database manager configuration parameter is set to yes, because when **intra_parallel** is set to no, there will be no shared sorts.
- If the concentrator is on (that is, when **max_connections** is greater than **max_coordagents**), because sorts that use a cursor declared with the WITH HOLD option will be allocated from shared memory.

smtp_server - SMTP server

This parameter identifies a simple mail transfer protocol (SMTP) server. This SMTP server transmits email sent by the UTL_MAIL built-in module.

The parameter also accepts a comma-delimited list of SMTP servers. UTL_MAIL attempts to send email through the first SMTP server in the list. If that SMTP server is unavailable, the next server in the list is used. If all servers in the comma-delimited list are unreachable, an error is returned.

Configuration type
Database

Applies to
Database server with local and remote clients

Database server with local clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
Null [comma-delimited list of valid SMTP server TCP/IP hostnames]

softmax - Recovery range and soft checkpoint interval

This parameter determines the frequency of soft checkpoints and the recovery range, which help out in the crash recovery process.

Configuration Type
Database

Parameter Type
Configurable

Default [range]

DB2 pureScale environment
100 [1 - 65 535]

Outside of a DB2 pureScale environment
100 [1 - 100 * **logprimary**]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of Measure
Percentage of the size of one primary log file

This parameter is used to:

- Influence the number of log files that need to be recovered following a crash (such as a power failure). For example, if the default value of 100 is used, the database manager will try to keep the number of log files that need to be recovered to 1. If you specify 300 as the value of this parameter, the database manager will try to keep the number of log files that need to be recovered to 3.

  To influence the number of log files required for crash recovery, the database manager uses this parameter to trigger the page cleaners to ensure that pages older than the specified recovery window are already written to disk.
- Determine the frequency of soft checkpoints. It is the process of writing information to the log control file. This information is used to determine the starting point in the log in case a database restart is required.

At the time of a database failure resulting from an event such as a power failure, there might have been changes to the database which:

- Have not been committed, but updated the data in the buffer pool
- Have been committed, but have not been written from the buffer pool to the disk
- Have been committed and written from the buffer pool to the disk.

When a database is restarted, the log files will be used to perform a crash recovery of the database which ensures that the database is left in a consistent state (that is, all committed transactions are applied to the database and all uncommitted transactions are not applied to the database).

To determine which records from the log file need to be applied to the database, the database manager uses information recorded in a log control file. (The database manager actually maintains two copies of the log control file, SQLOGCTL.LFH.1 and

SQLOGCTL.LFH.2, so that if one copy is damaged, the database manager can still use the other copy.) These log control files are periodically written to disk, and, depending on the frequency of this event, the database manager might be applying log records of committed transactions or applying log records that describe changes that have already been written from the buffer pool to disk. These log records have no impact on the database, but applying them introduces some additional processing time into the database restart process.

The log control files are always written to disk when a log file is full, and during soft checkpoints. You can use this configuration parameter to control the frequency of soft checkpoints.

The timing of soft checkpoints is based on the difference between the "current state" and the "recorded state", given as a percentage of the **logfilsiz**. The "recorded state" is determined by the oldest valid log record indicated in the log control files on disk, while the "current state" is determined by the log control information in memory. (The oldest valid log record is the first log record that the recovery process would read.) The soft checkpoint will be taken if the value calculated by the following formula is greater than or equal to the value of this parameter:

```
( (space between recorded and current states) / logfilsiz ) * 100
```

Recommendation: You might want to increase or reduce the value of this parameter, depending on whether your acceptable recovery window is greater than or less than one log file. Lowering the value of this parameter will cause the database manager both to trigger the page cleaners more often and to take more frequent soft checkpoints. These actions can reduce both the number of log records that need to be processed and the number of redundant log records that are processed during crash recovery.

Note however, that more page cleaner triggers and more frequent soft checkpoints increase the processing time associated with database logging, which can impact the performance of the database manager. Also, more frequent soft checkpoints might not reduce the time required to restart a database, if you have:

- Very long transactions with few commit points.
- A very large buffer pool and the pages containing the committed transactions are not written back to disk very frequently. (Note that the use of asynchronous page cleaners can help avoid this situation.)

In both of these cases, the log control information kept in memory does not change frequently and there is no advantage in writing the log control information to disk, unless it has changed.

sortheap - Sort heap size

This parameter defines the maximum number of private or shared memory pages that an operation requiring sort heap memory will consume.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]

32-bit platforms
Automatic [16 - 524 288]

64-bit platforms
Automatic [16 - 4 194 303]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
As needed to perform operations requiring sort memory

When freed
When operations requiring sort memory are complete

If the sort is a private sort, then this parameter affects agent private memory. If the sort is a shared sort, then this parameter affects the database shared memory. Each sort has a separate sort heap that is allocated as needed, by the database manager. This sort heap is the area where data is sorted. If directed by the optimizer, a smaller sort heap than the one specified by this parameter is allocated using information provided by the optimizer.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change.

The value of **sortheap** is tuned together with the **sheapthres_shr** parameter, therefore disabling self tuning of the **sortheap** parameter can not be done without disabling self tuning of the **sheapthres_shr** parameter. Enabling self tuning of the **sheapthres_shr** parameter automatically enables self tuning of the **sortheap** parameter. The **sortheap** parameter can, however, be enabled for self tuning without the **sheapthres_shr** parameter being AUTOMATIC.

Automatic tuning of **sortheap** is allowed only when the database manager configuration parameter **sheapthres** is set to 0.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the **self_tuning_mem** configuration parameter is set to ON.)

Recommendation: When working with the sort heap, you should consider the following factors:

- Appropriate indexes can minimize the use of the sort heap.
- The following use sort heap memory. Increase the size of this parameter when any of these techniques are used:
  - hash join buffers
  - block index ANDing
  - merge joins
  - table in memory
  - dynamic bitmaps (used for index ANDing and Star Joins)
  - table queues (when there are multiple database agents processing a query)

  - partial early distinct operations
  - partial early aggregation operations
- Increase the size of this parameter when frequent large sorts are required.
- When increasing the value of this parameter, you should examine whether the **sheapthres** and **sheapthres_shr** parameters in the database manager configuration file also need to be adjusted.
- The sort heap size is used by the optimizer in determining access paths. You should consider rebinding applications (using the **REBIND** command) after changing this parameter.

When the **sortheap** value is updated, the database manager will immediately start using this new value for any current or new sorts.

sql_ccflags - Conditional compilation flags

This parameter contains a list of conditional compilation values for use in conditional compilation of selected SQL statements.

Configuration type
Database

Parameter type
Configurable Online

The value of **sql_ccflags** must include one or more name and value pairs, where the name is separated from the value by the colon character. Each name and value pair must be separated from the previous pair by a comma. The name must be a valid ordinary SQL identifier. The value must be an SQL BOOLEAN constant, an SQL INTEGER constant, or the NULL keyword. The maximum length of the string is 1023 bytes.

When the value of **sql_ccflags** is updated, the value is not immediately checked to ensure that it is valid. Checking occurs the first time that the value is used to initialize the CURRENT SQL_CCFLAGS special register; that is, when a connection to the database first references CURRENT SQL_CCFLAGS, or when an inquiry directive is encountered in an SQL statement. After updating the value of **sql_ccflags**, connect to the database and query the special register by using the following statement: `VALUES CURRENT SQL_CCFLAGS`.

stat_heap_sz - Statistics heap size

This parameter indicates the *maximum* size of the heap used in collecting statistics using the **RUNSTATS** command.

The constraint set by this parameter applies to each **RUNSTATS** operation.

With Version 9.5, this database configuration parameter has a default value of `AUTOMATIC`, meaning that it increases as needed until either the **appl_memory** limit is reached, or the **instance_memory** limit is reached.

Configuration type
Database

Parameter type
- Configurable online
- Configurable by member in a DB2 pureScale environment

Default [range]
Automatic [1 096 - 524 288]

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
When the **RUNSTATS** utility is started

When freed
When the **RUNSTATS** utility is completed

Recommendation: **RUNSTATS** memory requirements depend on several factors. More memory is required with more statistic options, for example if LIKE statistics or DETAILED index statistics are being collected. When column statistics are collected, gathering statistics of a higher number of columns will require more memory. When distribution statistics are being collected, gathering a higher number of frequent, quantile values, or both will require more memory. The default setting of AUTOMATIC is recommended.

stmt_conc - Statement concentrator configuration parameter

This configuration parameter sets the default statement concentrator behavior.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Statement boundary

Default [range]
OFF [OFF, LITERALS]

This configuration parameter enables statement concentration for dynamic statements. The setting in the database configuration is used only when the client does not explicitly enable or disable the statement concentrator.

When enabled, the statement concentrator modifies dynamic statements to allow increased sharing of package cache entries.

The statement concentrator is disabled when the configuration parameter is set to OFF. When the configuration parameter is set to LITERALS, the statement concentrator is enabled. When the statement concentrator is enabled, SQL statements that are identical, except for the values of literals in the statements, might share package cache entries.

For example, when **stmt_conc** is set to LITERALS, the following statements share an entry in the package cache

```
SELECT FIRSTNME, LASTNAME FROM EMPLOYEE WHERE EMPNO='000020'
SELECT FIRSTNME, LASTNAME FROM EMPLOYEE WHERE EMPNO='000070'
```

The entry in the package cache uses the following statement:

```
SELECT FIRSTNME, LASTNAME FROM EMPLOYEE WHERE EMPNO=:L0
```

The DB2 database system provides the value for `:L0` based on the literal used in the original statements:

```
:L0(either '000020' or '000070')
```

This parameter can have a significant impact on access plan selection because it alters the statement text. The statement concentrator must be used only when similar statements in the package cache have similar plans. For example, if different literal values in a statement result in different plans, then statement concentrator must not be set to `LITERALS`.

The **stmt_conc** configuration parameter might cause the length attributes for VARCHAR and VARGRAPHIC string literals to be greater than the length of the string literal.

stmtheap - Statement heap size

This parameter specifies the size of the statement heap, which is used as a work space for the SQL or XQuery compiler during compilation of an SQL or XQuery statement.

Configuration type
Database

Parameter type
Configurable Online

Configurable by member in a DB2 pureScale environment

Propagation class
Statement boundary

Default [range]

For 32-bit platforms
`AUTOMATIC [128 - 524288]`

- Database server with local and remote clients: the default value is `AUTOMATIC` with an underlying value of `2048`.
- This parameter can also be set to a fixed value only, without the AUTOMATIC attribute.

For 64-bit platforms
`AUTOMATIC [128 - 524288]`

- Database server with local and remote clients: the default value is `AUTOMATIC` with an underlying value of `8192`.
- This parameter can also be set to a fixed value only, without the AUTOMATIC attribute.

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
For each statement during precompiling or binding

When freed
When precompiling or binding of each statement is complete

This area does not stay permanently allocated, but is allocated and released for every SQL or XQuery statement handled. Note that for dynamic SQL or XQuery statements, this work area will be used during execution of your program; whereas, for static SQL or XQuery statements, it is used during the bind process but not during program execution.

The **stmtheap** parameter can be set to AUTOMATIC with an underlying value or to a fixed value. When it is set to AUTOMATIC, the underlying value enforces a limit on the amount of memory allocated for a single compilation using dynamic join enumeration. If a memory limit is encountered, the statement compilation restarts using greedy join enumeration and an unlimited statement heap. It is only limited by the amount of remaining application memory (**appl_memory**), instance memory (**instance_memory**), or system memory. If greedy join enumeration completes successfully, an SQL0437W (reason code 1) warning will be returned to the application. If greedy join enumeration also encounters a memory limit, the statement preparation will fail with an SQL0101N error.

For example, **db2 update db cfg for SAMPLE using STMTHEAP 8192 AUTOMATIC** results in a statement heap limit of 8192 * 4K (32MB) for dynamic join enumeration and unlimited for greedy join enumeration.

When the **stmtheap** parameter is set to a fixed value, the limit applies to both dynamic and greedy join enumeration. If dynamic join enumeration encounters a memory limit, greedy join enumeration is attempted with the same fixed statement heap limit. Similar warnings/errors apply as in the AUTOMATIC case.

For example, **db2 update db cfg for SAMPLE using STMTHEAP 8192** results in a statement heap limit of 8192 * 4K (32MB) for both dynamic and greedy join enumeration.

If the runtime performance of your query is not sufficient, consider increasing the **stmtheap** configuration parameter value (either the value underlying AUTOMATIC or a fixed value) to ensure that dynamic programming join enumeration is successful. If you update the **stmtheap** configuration parameter to improve the performance of a query, cause the statement to be recompiled so that the query optimizer may create a new access plan to take advantage of the changed amount of statement heap.

Note: Dynamic programming join enumeration occurs only at optimization classes 3 and higher (5 is the default).

suspend_io - Database I/O operations state configuration parameter

This parameter shows whether the I/O write operations for a database are suspended or are being suspended.

Configuration type
Database

Parameter type
Informational

Values are YES, IN_PROGRESS, and NO.

systime_period_adj - Adjust temporal SYSTEM_TIME period database configuration parameter

This database configuration parameter specifies what action to take when a history row for a system-period temporal table is generated with an end timestamp that is less then the begin timestamp.

This can happen when two different transactions conflict in their attempt to update the same row in a system-period temporal table. It can also happen as the result of an adjustment to the system clock; for example, if the system clock is adjusted back an hour for the end of daylight savings time.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable online

Propagation class
Immediate

Default [range]
NO [NO, YES]

When a row is updated in a system-period temporal table, a history row is generated with a SYSTEM_TIME period indicating the range of time when the data in the history row was current. The value in the row-begin column indicates when the data in the history row became current. The value in the row-end column indicates when the history row became history data.

Following is an example of how two conflicting transactions could potentially generate a history row that has an row-end timestamp that is less than the row-begin timestamp.

1. Transaction TRA has a row-begin value generated for a row in a system-period temporal table it is updating at timestamp T1.
2. Transaction TRB has a row-begin value generated for the same row that it is updating at timestamp T2 (where T1 < T2).
3. Transaction TRB generates a history row and commits.
4. Transaction TRA generates its history row and commits.

After this sequence of events, the history row generated for transaction TRA would have an end timestamp that is less than its begin timestamp.

The database manager can ensure that generated history rows always have an end timestamp greater than the start timestamp by allowing timestamp adjustments when there are conflicts or by rolling back one of the transactions involved.

NO No timestamp value adjustments are made when the end timestamp is less than the start timestamp for a history row that is being inserted. Instead, the transaction that is attempting to insert the history row fails and an error is returned (SQLSTATE 57062, SQLCODE SQL20528N). Not allowing

adjustments ensures that all history rows generated during the transaction have the same end timestamp and can easily be identified using that end timestamp.

YES An adjustment is made to the timestamp value of the row-begin column value for the system-period temporal table and the end timestamp value for the generated history row when there are timestamp conflicts. The adjustment consists of modifying the end timestamp to be greater than the start timestamp by 1 microsecond. This ensures that the end timestamp is greater than the start timestamp for the history row. A message is returned indicating that an adjustment was made (SQLSTATE 01695, SQLCODE SQL5191W).

When no timestamp adjustments are necessary, SQLCODE DB20000I is returned.

Application programmers might consider using SQLCODE or SQLSTATE values to handle these timestamp value adjustment-related return codes from SQL statements.

territory - Database territory

This parameter shows the territory used to create the database. **territory** is used by the database manager when processing data that is territory sensitive.

Configuration type
Database

Parameter type
Informational

trackmod - Track modified pages enable

This parameter specifies whether the database manager will track database modifications so that the backup utility can detect which subsets of the database pages must be examined by an incremental backup and potentially included in the backup image.

Configuration type
Database

Parameter type
Configurable

Default [range]
No [Yes, No]

After setting this parameter to "Yes", you must take a full database backup in order to have a baseline against which incremental backups can be taken. Also, if this parameter is enabled and if a table space is created, then a backup must be taken which contains that table space. This backup could be either a database backup or a table space backup. Following the backup, incremental backups will be permitted to contain this table space.

tsm_mgmtclass - Tivoli Storage Manager management class

The Tivoli Storage Manager management class determines how the TSM server should manage the backup versions of the objects being backed up.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Default [range]
Null [any string]

The default is that there is no DB2 specified management class.

When performing any TSM backup, before using the management class specified by the database configuration parameter, TSM first attempts to bind the backup object to the management class specified in the INCLUDE-EXCLUDE list found in the TSM client options file. If a match is not found, the default TSM management class specified on the TSM server will be used. TSM will then rebind the backup object to the management class specified by the database configuration parameter.

Thus, the default management class, as well as the management class specified by the database configuration parameter, must contain a backup copy group, or the backup operation will fail.

tsm_nodename - Tivoli Storage Manager node name

This parameter is used to override the default setting for the node name associated with the Tivoli Storage Manager (TSM) product.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Statement boundary

Default [range]
Null [any string]

The node name is needed to allow you to restore a database that was backed up to TSM from another node.

The default is that you can only restore a database from TSM on the same node from which you did the backup. It is possible for the **tsm_nodename** to be overridden during a backup done through DB2 (for example, with the **BACKUP DATABASE** command).

tsm_owner - Tivoli Storage Manager owner name

This parameter is used to override the default setting for the owner associated with the Tivoli Storage Manager (TSM) product.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Statement boundary

Default [range]
Null [any string]

The owner name is needed to allow you to restore a database that was backed up to TSM from another node. It is possible for the **tsm_owner** to be overridden during a backup done through DB2 (for example, with the **BACKUP DATABASE** command).

Note: The owner name is case sensitive.

The default is that you can only restore a database from TSM on the same node from which you did the backup.

tsm_password - Tivoli Storage Manager password

This parameter is used to override the default setting for the password associated with the Tivoli Storage Manager (TSM) product.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Statement boundary

Default [range]
Null [any string]

The password is needed to allow you to restore a database that was backed up to TSM from another node.

Note: If the **tsm_nodename** is overridden during a backup done with DB2 (for example, with the **BACKUP DATABASE** command), the **tsm_password** might also have to be set.

The default is that you can only restore a database from TSM on the same node from which you did the backup. It is possible for the **tsm_nodename** to be overridden during a backup done with DB2.

user_exit_status - User exit status indicator

If set to YES, the **user_exit_status** parameter indicates that the database manager is enabled for rollforward recovery and that the database archives and retrieves log files based on the values set by either the **logarchmeth1** parameter or the **logarchmeth2** parameter.

Configuration type
Database

Parameter type
Informational

util_heap_sz - Utility heap size

This parameter indicates the maximum amount of memory that can be used simultaneously by the backup, restore, and load (including load recovery) utilities.

Configuration type
Database

Parameter type

- Configurable online
- Configurable by member in a DB2 pureScale environment

Propagation class
Immediate

Default [range]
`5000 [16 - 524 288 ]`

Note: The default value is subject to change by the DB2 Configuration Advisor after initial database creation.

Unit of measure
Pages (4 KB)

When allocated
As required by the database manager utilities

When freed
When the utility no longer needs the memory

Recommendation: Use the default value unless your utilities run out of space, in which case you should increase this value. If memory on your system is constrained, you might want to lower the value of this parameter to limit the memory used by the database utilities. If the parameter is set too low, you might not be able to run utilities concurrently. You should update this parameter dynamically as needed. For a small number of utilities, set this parameter to a small value. For a large number of utilities, or for memory intensive utilities, you should set this parameter to a larger value.

varchar2_compat - varchar2 compatibility database configuration parameter

This parameter indicates whether the compatibility semantics associated with the VARCHAR2 and NVARCHAR2 data types are applied to the connected database.

Configuration type
Database

Parameter type
Informational

The value is determined at database creation time, and is based on the setting of the **DB2_COMPATIBILITY_VECTOR** registry variable for VARCHAR2 support. The value cannot be changed.

vendoropt - Vendor options

This parameter specifies additional parameters that DB2 might need to use to communicate with storage systems during backup, restore, or load copy operations.

Configuration type
Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Default [range]
Null [ ]

Restrictions
You cannot use the **vendoropt** configuration parameter to specify vendor-specific options for snapshot backup or restore operations. You must use the **OPTIONS** parameter of the backup or restore utilities instead.

In TSM environments configured to support proxy nodes, the `"-fromnode=nodename"` option and the `"-fromowner=ownername"` option are not compatible with the `"-asnodename=nodename"` option and cannot be used together. Use the `-asnodename` option for TSM configurations using proxy nodes and the other two options for other types of TSM configurations. For more information, see "Configuring a Tivoli Storage Manager client".

wlm_collect_int - Workload management collection interval configuration parameter

This parameter specifies a collect and reset interval, in minutes, for workload management (WLM) statistics.

Every *x* minutes, (where *x* is the value of the **wlm_collect_int** parameter) all workload management statistics are collected and sent to any active statistics event monitor; then the statistics are reset. If an active statistics event monitor exists, depending on how it was created, the statistics are written to a file, to a pipe, or to a table. If it does not exist, the statistics are only reset and not collected.

Collections occur at the specified interval times as measured relative to Sunday at 00:00:00. When the catalog member becomes active, the next collection will occur at the start of the next scheduled interval relative to this fixed time. The scheduled interval is not relative to the catalog member activation time. If a member is not active at the time of collection, no statistics are gathered for that member. For example, if the interval value was set to 60 and the catalog member was activated on 9:24 AM on Sunday, then the collections would be scheduled to occur each hour on the hour. Therefore, the next collection will occur at 10:00 AM. If the member is not active at 10:00 AM, then no statistics will be gathered for that member.

The collect and reset process is initiated from the catalog member. The **wlm_collect_int** parameter must be specified on the catalog member. It is not used on other members.

Configuration type
Database

Parameter type

- Configurable online

Default [range]
`0` [`0` (no collection performed), `5 - 32 767`]

The workload management statistics collected by a statistics event monitor can be used to monitor both short term and long term system behavior. A small interval can be used to obtain both short term and long term system behavior because the results can be merged together to obtain long term behavior. However, having to manually merge the results from different intervals complicates the analysis. If it's not required, a small interval unnecessarily increases the processing time. Therefore, reduce the interval to capture shorter term behavior, and increase the interval to reduce processing time when only analysis of long term behavior is sufficient.

The interval needs to be customized per database, not for each SQL request, or command invocation, or application. There are no other configuration parameters that need to be considered.

Note: All WLM statistics table functions return statistics that have been accumulated since the last time the statistics were reset. The statistics will be reset regularly on the interval specified by this configuration parameter.

DB2 Administration Server (DAS) configuration parameters

authentication - Authentication type DAS

This parameter determines how and where authentication of a user takes place.

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 .

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable

Default [range]
SERVER_ENCRYPT [SERVER_ENCRYPT; KERBEROS_ENCRYPT]

If **`authentication`** is SERVER_ENCRYPT, then the user ID and password are sent from the client to the server so authentication can take place on the server. User IDs and passwords sent over the network are encrypted.

A value of KERBEROS_ENCRYPT means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication.

Note: The KERBEROS_ENCRYPT authentication type is only supported on servers running Windows.

This parameter can only be updated from a Version 9 command line processor (CLP).

contact_host - Location of contact list

This parameter specifies the location where the contact information used for notification by the Scheduler and the Health Monitor is stored.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
Null [any valid DB2 administration server TCP/IP hostname]

The location is defined to be a DB2 administration server's TCP/IP hostname. Allowing **contact_host** to be located on a remote DAS provides support for sharing a contact list across multiple DB2 administration servers. If **contact_host** is not specified, the DAS assumes the contact information is local.

This parameter can only be updated from a Version 8 command line processor (CLP).

das_codepage - DAS code page

This parameter indicates the code page used by the DB2 administration server.

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 .

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
Null [any valid DB2 code page]

If the parameter is null, then the default code page of the system is used. This parameter should be compatible with the locale of the local DB2 instances. Otherwise, the DB2 administration server cannot communicate with the DB2 instances.

This parameter can only be updated from a Version 8 command line processor (CLP).

das_territory - DAS territory

This parameter shows the territory used by the DB2 administration server.

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 .

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
Null [any valid DB2 territory]

If the parameter is null, then the default territory of the system is used.

This parameter can only be updated from a Version 8 command line processor (CLP).

dasadm_group - DAS administration authority group name

This parameter defines the group name with DAS Administration (DASADM) authority for the DAS.

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 .

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable

Default [range]
Null [any valid group name]

DASADM authority is the highest level of authority within the DAS.

DASADM authority is determined by the security facilities used in a specific operating environment.

- For the Windows operating systems, this parameter can be set to any local group that is defined in the Windows security database. Group names are accepted as long as they are 30 bytes or less in length. If "NULL" is specified for this parameter, all members of the Administrators group have DASADM authority.
- For Linux and UNIX systems, if "NULL" is specified as the value of this parameter, the DASADM group defaults to the primary group of the instance owner.

  If the value is not "NULL", the DASADM group can be any valid UNIX group name.

db2system - Name of the DB2 server system

This parameter specifies the name that is used by your users and database administrators to identify the DB2 server system.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable Online

Default [range]
TCP/IP host name [any valid system name]

If possible, this name should be unique within your network.

This name aids users in identifying the system that contains the database they want to access. A value for **db2system** is set at installation time as follows:

- On Windows, the **setup** program sets it equal to the computer name specified for the Windows system.
- On UNIX systems, it is set equal to the UNIX system's TCP/IP hostname.

diaglevel - Diagnostic error capture level configuration parameter

This parameter specifies the type of diagnostic errors that will be recorded in the `db2dasdiag.log` file.

Configuration Type
Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type
Configurable Online

Propagation Class
Immediate

Default [Range]
`3 [0 - 4]`

Valid values for this parameter are:

- `0` - No diagnostic data captured
- `1` - Severe errors only
- `2` - All errors
- `3` - All errors and warnings
- `4` - All errors, warnings, and informational messages

Usage notes

- The dynamic behavior for **diaglevel** does not extend to all processes.
- The **db2sysc** DB2 server process can detect dynamic changes, for example, when you issue the **UPDATE DATABASE MANAGER CONFIGURATION** command over an instance attachment.
- When DB2 client and application processes start, they use the **diaglevel** configuration parameter setting and do not detect any dynamic changes.
- To help resolve a problem, you can increase the value of this parameter to gather additional problem determination data.
- In specific circumstances, to display high importance messages, DB2 will override the **diaglevel** configuration parameter setting.

discover - DAS discovery mode

This parameter determines the type of discovery mode that is started when the DB2 Administration Server starts.

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 .

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
`SEARCH [DISABLE; KNOWN; SEARCH ]`

- If **discover** = `SEARCH`, the administration server handles SEARCH discovery requests from clients. SEARCH provides a superset of the functionality provided by KNOWN discovery. When **discover** = `SEARCH`, the administration server will handle both SEARCH and KNOWN discovery requests from clients.
- If **discover** = `KNOWN`, the administration server handles only KNOWN discovery requests from clients.
- If **discover** = `DISABLE`, then the administration server will not handle any type of discovery request. The information for this server system is essentially hidden from clients.

Discovery mode is enabled by default.

exec_exp_task - Execute expired tasks

This parameter specifies whether or not the Scheduler will execute tasks that have been scheduled in the past, but have not yet been executed.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable

Default [range]
`No [Yes; No ]`

The Scheduler only detects expired tasks when it starts up. For example, if you have a job scheduled to run every Saturday, and the Scheduler is turned off on Friday and then restarted on Monday, the job scheduled for Saturday is now a job that is scheduled in the past. If **exec_exp_task** is set to `Yes`, your Saturday job will run when the Scheduler is restarted.

jdk_path - Software Developer's Kit for Java installation path DAS

This parameter specifies the directory under which the Software Developer's Kit (SDK) for Java, to be used for running DB2 administration server functions, is installed.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
Default Java install path [any valid path]

Environment variables used by the Java interpreter are computed from the value of this parameter.

On Windows operating systems, Java files (if needed) are placed under the `sqllib` directory (in `java\jdk`) during DB2 installation. The **jdk_path** configuration parameter is then set to `sqllib\java\jdk`. Java is never actually installed by DB2 on Windows platforms; the files are merely placed under the `sqllib` directory, and this is done regardless of whether or not Java is already installed.

This parameter can only be updated from a Version 8 command line processor (CLP).

sched_enable - Scheduler mode

This parameter indicates whether or not the Scheduler is started by the administration server.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable

Default [range]
Off [On; Off]

The Scheduler allows tools such as the Data Studio to schedule and execute tasks at the administration server.

This parameter can only be updated from a Version 8 command line processor (CLP).

sched_userid - Scheduler user ID

This parameter specifies the user ID used by the Scheduler to connect to the tools catalog database. This parameter is only relevant if the tools catalog database is remote to the DB2 administration server.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Informational

Default [range]
Null [any valid user ID]

The userid and password used by the Scheduler to connect to the remote tools catalog database are specified using the **db2admin** command.

smtp_server - SMTP server

When the Scheduler is on, this parameter identifies the SMTP server that the Scheduler will use to send email and pager notifications.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
Null [any valid SMTP server TCP/IP hostname]

This parameter is used by the Scheduler and the Health Monitor.

This parameter can only be updated from a Version 8 command line processor (CLP).

toolscat_db - Tools catalog database

This parameter indicates the tools catalog database used by the Scheduler.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable

Default [range]
Null [any valid database alias]

This database must be in the database directory of the instance specified by **toolscat_inst**.

toolscat_inst - Tools catalog database instance

This parameter indicates the instance name that is used by the Scheduler, along with toolscat_db and toolscat_schema, to identify the tools catalog database.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable

Default [range]
Null [any valid instance]

The tools catalog database contains task information. The tools catalog database must be listed in the database directory of the instance specified by this configuration parameter. The database can be local or remote. If the tools catalog database is local, the instance must be configured for TCP/IP. If the database is remote, the database partition cataloged in the database directory must be a TCP/IP node.

toolscat_schema - Tools catalog database schema

This parameter indicates the schema of the tools catalog database used by the Scheduler.

Configuration type
DB2 Administration Server

Applies to
DB2 Administration Server

Parameter type
Configurable

Default [range]
Null [any valid schema]

The schema is used to uniquely identify a set of tools catalog tables and views within the database.

This parameter can only be updated from a Version 8 command line processor (CLP).

cf_sca_sz - Shared communication area configuration parameter

This Shared Communication Area (SCA) configuration parameter determines the memory size used by the SCA in the cluster caching facility (CF). The SCA is a per database entity and contains database wide control block information for tables, indexes, table spaces, and catalogs.

Configuration type
Database

Applies to
DB2 pureScale environment

Parameter type
Configurable online

Default [range]
AUTOMATIC [AUTOMATIC, 2048 - 1073741824]

Unit of measure
Pages (4 KB)

When allocated
When the database is first activated on any DB2 member

When freed
When the database is dropped or when the CF server is stopped

When you set the **cf_sca_sz** parameter to AUTOMATIC (default), the memory value is computed by the DB2 database manager during the first database activation on any member to a value that is sufficient for basic database operations. If you want to configure the exact memory size for the SCA, you can set the **cf_sca_sz** parameter to a fixed numeric value.

You can obtain the current value for the SCA size by running the **GET DB CFG SHOW DETAIL** command. If the **cf_sca_sz** parameter is set to AUTOMATIC, the SHOW DETAIL clause displays the computed value as well as the allocated value for the SCA.

If the SCA memory is fully consumed by database operations, an out of memory error message is returned. In this case, you can increase the size of the SCA by setting a higher value for the **cf_sca_sz** parameter.

See “Configuring cluster caching facility memory for a database” for more detail.

DB2 pureScale Feature cluster caching facility configuration

In a DB2 pureScale environment, the cluster caching facility, also known as CF, is used to facilitate global locking and buffer pool management. CF configuration is integrated into DB2 infrastructure and managed by DB2 configuration interfaces.

The CF configuration has a unique set of configuration parameters to optimize a DB2 pureScale environment.

Configuration parameters for the CF are categorized as either CF server configuration parameters or CF structure configuration parameters. CF server parameters are configured as DB2 database manager configuration parameters. CF

structure parameters are configured as DB2 database configuration parameters. Default settings are applied to the configuration file when a DB2 pureScale Feature instance or a database is created.

Cluster caching facility configuration

The CF server configuration information is maintained as part of the DB2 database manager configuration information. As such, the CF server is configured in the same way the database manager is configured using DB2 configuration interfaces. The DB2 Command Line Processor (CLP) or the configuration APIs can be used to display and update configurable CF server information.

The following configuration parameters support the CF server:

- **cf_diaglevel**
- **cf_diagpath**
- **cf_mem_sz**
- **cf_num_conns**
- **cf_num_workers**

Cluster caching facility structure configuration

CF structure memory is maintained as part of DB2 database configuration information. CF structure parameters are configured in the same way DB2 database parameters are configured using the DB2 CLP or configuration APIs to display and update configurable CF structure.

The following configuration parameters are added to support the CF structure:

- **cf_db_mem_sz**
- **cf_gbp_sz**
- **cf_lock_sz**
- **cf_sca_sz**

CF structures are created in the CF server for each database. During a call to **CREATE DATABASE** command, the DB2 configuration advisor also sets CF structure configuration parameters to their default value AUTOMATIC and computes the appropriate startup values for your DB2 pureScale instance.

Proper CF structure configuration is important to achieve optimal performance in a DB2 pureScale environment. The computed defaults for CF structures might not achieve optimal performance for your DB2 pureScale environment. In a performance benchmark environment, manually tune the CF structure sizes to achieve the required level of performance. Also be aware that using table partitioning under DB2 pureScale incurs additional memory usage.

Configuration Advisor

The DB2 Configuration Advisor is invoked when creating a database to set a reasonable set of defaults for the database configuration based on the host environment such as the amount of memory and the number of processors.

The DB2 Configuration Advisor is enhanced with the DB2 pureScale Feature to compute the startup values for the CF structure configuration parameters. The parameters **cf_db_mem_sz**, **cf_gbp_sz**, **cf_lock_sz**, and **cf_sca_sz** are computed

based on the total CF memory size defined by **cf_mem_sz** to ensure the best distribution of memory among these structures to achieve reasonable performance standards for a DB2 pureScale instance.

High availability with two cluster caching facilities

You must set up two CF servers when running in a Highly Available (HA) CF environment. One CF acts as the primary server and the other as a backup or secondary CF server. In this environment the primary CF server and the secondary CF server share the same host configuration.

For more information about structure duplex support in high availability environments, refer to Structure duplex support behavior with a secondary cluster caching facility.

In a DB2 pureScale environment where there are two CF servers, you can replace both cluster caching facilities using a rolling upgrade technique to maintain an environment that is highly available.

For more information about maintaining a high availability environment during CF replacement, refer to Replacing both cluster caching facilities.

Configuring the cluster caching facility

Use the DB2 command line processor (CLP) or the DB2 configuration APIs to configure the cluster caching facility (CF) parameters for the IBM DB2 pureScale Feature.

About this task

The cluster caching facility configuration information is maintained as part of the DB2 database manager configuration information.

When the cluster caching facility is started, CF server configuration parameters are read from the database manager configuration file to initialize the CF server. The CF server creates and listens on a management port to receive CF server configuration information. Once a **db2start** command or an online action by a CF server or CF structure is finished, the CF server receives the server configuration parameters, and is fully initialized.

If the CF server configuration parameters are set to `AUTOMATIC`, their values are determined automatically during the startup of the CF server. During this startup, the CF server host is queried for the amount of physical RAM and the number of CPUs; this helps configure the appropriate values for the **cf_mem_sz** and **cf_num_workers** configuration parameters.

There are two ways to configure the CF server:

- Updating configuration parameters using the CLP
- Updating configuration parameters using APIs

Procedure

- To view the values of CF server configuration parameters in the CLP, issue the **GET DATABASE MANAGER CONFIGURATION** command.

  For example, the following CF server configuration information is displayed when this command is issued in a DB2 pureScale instance:

```
CF Server Configuration:
   Memory size (4KB)                    (CF_MEM_SZ) = AUTOMATIC
   Number of workers threads        (CF_NUM_WORKERS) = AUTOMATIC
   Number of connections              (CF_NUM_CONNS) = AUTOMATIC
   Diagnostic error capture level     (CF_DIAGLEVEL) = 2
   Diagnostic data directory path      (CF_DIAGPATH)
```

See the **GET DATABASE MANAGER CONFIGURATION** command for more detailed output.

- To update the values of CF server configuration parameters in the CLP, issue the **UPDATE DATABASE MANAGER CONFIGURATION** command.

  For example, the following information is displayed when this command is issued to update the CF memory size (**cf_mem_sz**):

```
UPDATE DBM CFG USING CF_MEM_SZ 27152

DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command
completed successfully.

SQL1362W  One or more of the parameters submitted for immediate
modification were not changed dynamically.  Client changes will
not be effective until the next time the application is started
or the TERMINATE command has been issued.  Server changes will
not be effective until the next DB2START command.
```

  The CF server configuration changes do not take effect immediately. Any updating value is saved and applied the next time the CF server is started.
- To reset the values of CF server configuration parameters in the CLP, issue the **RESET DATABASE MANAGER CONFIGURATION** (or **RESET DBM CFG**) command. The DB2 database manager configuration parameters are reset to their default value.
- To update the values of CF server configuration parameters using application programming interfaces (APIs), call the db2CfgSet API. To view the values, issue the db2CfgGet API.

  The getting and setting of CF server configuration parameters is supported by entries for the token element in the **db2Cfg** data structures used when calling the db2CfgGet or db2CfgSet APIs.

*Table 138. Supported **db2Cfg** tokens in getting and setting of CF server configuration parameters*

Token	Data type
SQLF_KTN_CF_MEM_SZ	Uint32
SQLF_KTN_CF_NUM_WORKERS	Uint32
SQLF_KTN_CF_NUM_CONNS	Uint32
SQLF_KTN_CF_DIAGLEVEL	Uint16
SQLF_KTN_CF_DIAGPATH SQLF_KTN_CF_DIAGPATH_FULL	char(215)

Configuring cluster caching facility memory for a database

Use the DB2 command line processor (CLP) or the DB2 configuration APIs to display and update cluster caching facility memory configuration parameters for the database.

About this task

The cluster caching facility structure memory configuration information is maintained as part of the DB2 database configuration information.

The cluster caching facility structure memory used for Group Buffer Pool (GBP), lock usage, and Shared Communication Area (SCA) is allocated for the cluster caching facility during the first database activation on any member and remains allocated until deactivation on the last member. These parameters have a default value set to AUTOMATIC. When set to AUTOMATIC, DB2 computes appropriate sizes for these parameters during database activation. Because these values are closely related and dependent on one another, manually setting at least one of the parameters causes none of the parameters to be calculated during activation even if some parameters remain set to AUTOMATIC. Their values are what the most recent automatically calculated value was and can be viewed with the **GET DB CFG SHOW DETAIL** command.

The ONLINE option is also supported for structure parameters. Any updates to CF memory parameters are applied immediately. Update requests are synchronous and are not returned until the new value is set by the CF server.

There are two ways to configure the cluster caching facility memory for a database:

- Updating configuration parameters using the CLP
- Updating configuration parameters using APIs

Procedure

- To view the values of the CF structure configuration parameters in the CLP, issue the **GET DATABASE CONFIGURATION** command.

  For example, the following CF structure configuration information is displayed when this command is issued in a DB2 pureScale instance:

```
CF Resource Configuration:
     CF database memory size (4KB)
              (CF_DB_MEM_SZ) = AUTOMATIC
     Group buffer pool size (4KB)
              (CF_GBP_SZ)    = AUTOMATIC
     Global lock memory size (4KB)
              (CF_LOCK_SZ)   = AUTOMATIC
     Shared communication area size (4KB)
              (CF_SCA_SZ)    = AUTOMATIC
```

  See the **GET DATABASE CONFIGURATION** command for more detailed output.

  Note: The group buffer pool (GBP) parameter requires a sufficient amount of directory entries in the CACHE structure to handle read-and-register (RAR) requests. DB2 monitors free directory entries and data element counts on every RAR request and automatically adjusts these values as needed. This adjustment is done dynamically or by online updates, regardless of whether the **cf_gbp_sz** parameter is configured as AUTOMATIC or not. For more information about new CF structure configuration parameters, see the topic "Configuration parameters summary for the DB2 pureScale Feature".

  When you use the **SHOW DETAIL** option to view the current values for structure configuration parameters, the following information is displayed:

```
CF Resource Configuration:
  Group buffer pool size (4KB)
              (CF_GBP_SZ)  = AUTOMATIC(32768)  AUTOMATIC(32768)
  Global lock memory size (4KB)
              (CF_LOCK_SZ) = AUTOMATIC(17000)  AUTOMATIC(17000)
  Shared communication area size (4KB)
              (CF_SCA_SZ)  = AUTOMATIC(1500)   AUTOMATIC(1500)
```

  See the **GET DATABASE CONFIGURATION** command for more detailed output.

  The result shown for each structure parameter has two corresponding values. If the values in parentheses are different there is a pending update to the value of

the structure parameter. The value enclosed in the first set of parentheses is the current value for the structure parameter. The value enclosed in the second set of parentheses is the pending value for the structure parameter.

When there is both a primary CF server and a secondary CF server, only the configuration information for the primary CF server is displayed. The same value applies to both CF servers.

- To update the values of CF structure configuration parameters in the CLP, issue the **UPDATE DATABASE CONFIGURATION** (or **UPDATE DB CFG**) command. Updating structure parameters to any value other than the default setting (AUTOMATIC) results in theCF structure memory size being fixed.

  For example, this command is issued to set the size of the Global Lock Manager memory to 20,000 pages:

  ```
  UPDATE DATABASE CONFIGURATION FOR WSDB USING CF_LOCK_SZ 20000
  ```

  Note: An update to decrease the amount of memory for structure parameters does not take immediate effect if there is not enough unused memory to accommodate the request. To see if an update is pending for structure parameters issue the **GET DB CFG** command from the CLP.
- To reset the values of CF structure configuration parameters in the CLP, issue the **RESET DATABASE CONFIGURATION** (or **RESET DB CFG**) command. The CF structure configuration parameters are reset to their default value of AUTOMATIC.
- To update the values of CF structure configuration parameters using application programming interfaces (APIs), call the db2CfgSet API. To view the values, issue the db2CfgGet API.

  The getting and setting of CF structure configuration parameters is supported by entries for the token element in the **db2Cfg** data structures used when calling the db2CfgGet or db2CfgSet APIs.

Table 139. Supported **db2Cfg** *tokens in getting and setting of CF structure configuration parameters*

Token	Data type
SQLF_DBTN_CF_GBP_SZ	Uint32
SQLF_DBTN_CF_SCA_SZ	Uint32
SQLF_DBTN_CF_LOCK_SZ	Uint32
SQLF_DBTN_CF_DB_MEM_SZ	Uint32

DB2 pureScale CF memory parameter configuration

In a DB2 pureScale instance, several configuration parameters and registry variables work together to control memory allocation for the cluster caching facility, also known as CF.

Parameters

The following configuration parameters and registry variables influence the CF memory allocation and management. These parameters and variables only come into effect after a member start or restart, except where noted in the description.

- **cf_mem_sz**: controls the total memory used by the CF. The default setting is AUTOMATIC.
- **cf_db_mem_sz**: controls the total memory limit used by the CF for this particular database. The default setting is AUTOMATIC.

- **cf_gbp_sz**: determines the memory used by the CF for group buffer pool usage, for this particular database. The default setting is AUTOMATIC.
- **cf_lock_sz**: determines the memory used by the CF for locking usage, for this particular database. The default setting is AUTOMATIC.
- **cf_sca_sz**: determines the memory used by the CF for the Shared Communication Area (SCA) for this particular database. The default setting is AUTOMATIC.
- **numdb**: specifies the number of local databases that can be concurrently active. The default number of active databases is 32. This parameter comes into effect only after a group restart.
- **DB2_DATABASE_CF_MEMORY**: this registry variable determines the percentage of the total CF memory assigned to each database where the **cf_db_mem_sz** value is set to AUTOMATIC. The default percentage is 100. This variable comes into effect only after a group restart.

When set to AUTOMATIC, the values of CF memory configuration parameters are dynamically generated based on the properties and attributes of the DB2 pureScale environment.

Parameter relationships

Note: Set the value of **numdb** higher than or equal to the total number of databases on the instance. This value includes active and inactive databases. Additionally, you can set the value of **DB2_DATABASE_CF_MEMORY** to a value so that every database on this instance, whether normally inactive or active, can be active at the same time.

The cluster caching facility memory parameters are defined at both an instance level and a database level:

- At the instance level, the total amount of memory reserved for the CF is determined by the **cf_mem_sz** configuration parameter.
- At the database level, the total amount of memory used by all the CF structures for a database is controlled by the **cf_db_mem_sz** configuration parameter.

The memory limit defined by the database-level **cf_db_mem_sz** parameter is constrained by the instance-level **cf_mem_sz** parameter. An upper limit is therefore placed on the **cf_db_mem_sz** parameter: it cannot exceed this memory threshold to accommodate the memory requirements for the CF database structures. The total sum value of the **cf_db_mem_sz** parameters for all active databases cannot exceed the value of **cf_mem_sz** for the instance.

Similarly, although the **cf_db_mem_sz** parameter controls the total amount of memory that can be used by all the CF structures for a database, at no time can the database manager reserve a block of memory equal to the value of **cf_db_mem_sz**.

The configuration parameters that control the amount of memory for each of the CF structures for a database are:

- **cf_gbp_sz**
- **cf_lock_sz**
- **cf_sca_sz**

The CF uses 3840 4k pages internally; these pages are taken from amount specified by the **cf_mem_sz** parameter. Additionally, there are 256 4k pages reserved from **cf_mem_sz** for every active database on the instance. These additional amounts are

only included when the **cf_mem_sz** parameter is set manually.

Automatic configurations

The default setting for the **cf_mem_sz** parameter is AUTOMATIC. This means that the amount of memory available for use by the CF is calculated using the total memory available on the CF host. The parameter is dynamically set to either:

- An appropriate percentage, typically 70% to 90%, of the total memory available on the CF host.
- The amount of free memory on the CF host.

whichever value is lower.

The default setting for the **cf_db_mem_sz** parameter is AUTOMATIC. The actual value is determined based on the **DB2_DATABASE_CF_MEMORY**, **cf_mem_sz**, and **numdb** parameter values in effect when the database is first activated on any member.

Another factor considered during AUTOMATIC computation is if the CF and any DB2 members coexist.

Online configurations

The **cf_mem_sz** parameter is not configurable online. The CF server must be restarted in order for a new value of **cf_mem_sz** to take effect.

The **cf_db_mem_sz** parameter is configurable online, however there are several constraints to note:

- The value of **cf_db_mem_sz** must always be less than the current value of **cf_mem_sz**. If you want to increase **cf_db_mem_sz** to a value greater than the current value of **cf_mem_sz**, you must first increase the value of **cf_mem_sz**.
- Even if the **cf_db_mem_sz** parameter is increased to a value that is less than the **cf_mem_sz** parameter, the following situations cause an error:
  - Requesting an online increase for a CF structure memory parameter, where the sum of the CF structure memory parameters does *not* exceed the value of **cf_db_mem_sz**, but *does* exceed the upper threshold limit set by the **cf_mem_sz** parameter. There is not enough available memory in the CF to satisfy the CF structure memory request, and an error is generated.
- The value of **cf_db_mem_sz** must always be greater than the sum of the **cf_gbp_sz**, **cf_lock_sz**, and **cf_sca_sz** parameters. If the value of **cf_db_mem_sz** is reduced, the values of those three CF structure parameters must also be reduced to accommodate the changed memory limit set by the **cf_db_mem_sz** parameter. Failure to do so results in an error because there is insufficient memory provided by the **cf_db_mem_sz** parameter.

The **cf_gbp_sz**, **cf_lock_sz**, and **cf_sca_sz** parameters are configurable online, however there are several constraints to note:

- Any attempt to change these parameters can result in a timeout error if the system is busy or there is not enough available CF memory at the time the request was submitted.

  Note: Although memory might not be available at the time the request is submitted, the system continues to attempt the update in case sufficient memory becomes available to satisfy the request. The update operation waits for the processing to be completed.

- During the update operation of a CF structure memory parameter, if a second request to update the same structure memory parameter is initiated, an error is triggered.

Multiple active databases

In a DB2 pureScale environment running multiple active databases, there are additional considerations when planning the memory configuration requirements for each active database. The key configuration parameter in this situation is **DB2_DATABASE_CF_MEMORY**. This parameter is used with the other CF configuration parameters, to indicate the percentage of the total CF memory that is allocated to each database. The **DB2_DATABASE_CF_MEMORY** parameter is expressed as a float data type, and if it has:

A value of -1
The dynamically generated value of the **cf_db_mem_sz** parameter for each active database is equal to the result of this calculation: **cf_mem_sz** divided by **numdb**:

$$cf\_db\_mem\_sz = \frac{cf\_mem\_sz}{numdb}$$

This means that every database that is active receives an equal share of memory from the **cf_mem_sz**, if that database has the **cf_db_mem_sz** parameter set to AUTOMATIC.

Note: If the calculated value of **cf_db_mem_sz** is smaller than the minimum value of the **cf_db_mem_sz** parameter, the **cf_db_mem_sz** value is automatically set to the minimum allowed value of **cf_db_mem_sz**.

A value of 0
Every database in the DB2 pureScale environment must have the **cf_db_mem_sz** value set manually. Otherwise, if the value of **cf_db_mem_sz** is set to AUTOMATIC, then that database receives the minimum allowed value of 32,768.

A value between 0 and 100
The value of the **cf_db_mem_sz** parameter is equal to the result of this calculation:

$$cf\_db\_mem\_sz = cf\_mem\_sz * \frac{db2\_database\_cf\_memory}{100}$$

Note: If the calculated value of **cf_db_mem_sz** is smaller than the minimum value of the **cf_db_mem_sz** parameter, the **cf_db_mem_sz** value is automatically set to the minimum allowed value of **cf_db_mem_sz**.

A value of 100
Only one database is able to activate in the DB2 pureScale environment, and it receives all of the CF memory defined by the **cf_mem_sz** parameter.

In configurations where **DB2_DATABASE_CF_MEMORY** is used to dynamically calculate the value for the **cf_db_mem_sz** parameter, the value of **cf_db_mem_sz** can never be smaller than its minimum allowed value, regardless of the setting of the **DB2_DATABASE_CF_MEMORY** parameter.

Restrictions

Self-tuning of CF memory between databases is not supported.

Examples

Example 1: In this example, the **DB2_DATABASE_CF_MEMORY** registry variable is not changed from the default setting of "100". Therefore, only one database receives all of the CF memory (an amount equal to the value defined by the **cf_mem_sz** parameter. If a second database is created or started, the command fails with an out of memory error. To avoid this situation, so that a second database can be created with equal memory sharing, the CF memory parameters must be modified (this assumes that **cf_db_mem_sz** is set to AUTOMATIC):

```
db2stop
db2 update dbm cfg using numdb 2
db2set db2_database_cf_memory=50
db2 start
```

Example 2: In this example, the **DB2_DATABASE_CF_MEMORY** registry variable is changed to give all databases an equal share of the CF memory, based on having two active databases:

```
db2stop
db2 update dbm cfg using numdb 2
db2set db2_database_cf_memory=-1
db2 start
db2 connect to INVOICES      # success
db2 connect to HRDEPT        # success
```

Example 3: In this example, the **DB2_DATABASE_CF_MEMORY** registry variable is changed to give all 200 databases an equal share of the CF memory, based on having 200 active databases:

```
db2stop
db2 update dbm cfg using numdb 200
db2set db2_database_cf_memory=0.5
db2 start
```

Structure duplex support behavior with a secondary cluster caching facility

When running in a highly available environment, you must use multiple cluster caching facilities to provide continuous support if acluster caching facility, also known as a CF, goes down.

The following structures are supported by the cluster caching facility:

- The group buffer pool (GBP)
- The CF lock manager (LOCK)
- The shared communication area (SCA).

The primary and secondary CF servers share the same host configuration in a highly available environment. It is recommended that host specifications such as CPU speed, number of CPUs, and the amount of random access memory (RAM) are the same for both CF servers. The two CF servers share one set of configuration parameters and access the same port numbers.

The CF has configurable structures, the GBP, LOCK, and SCA. Each structure supports online updates. When increasing the allotted memory for these structures, also known as growing a structure, the request is first sent to the secondary CF

server and then to the primary CF server. The reverse is true when decreasing the allotted memory for these structures, also known as shrinking a structure. The shrinking request is first sent to the primary CF server and then to the secondary CF server. A structure in the secondary CF is always as big or bigger than the structure in the primary CF to ensure there are no storage issues after a CF failover.

If either the primary CF or secondary CF fails to update a CF configuration parameter, the other CF rolls back to the previous setting for that CF configuration parameter. This rollback ensures that there is sufficient structure memory available on both CFs.

By default, a resize request waits until the resizing operation completes before returning to the client. If you interrupt the resize request with a user control action, such as **Ctrl+C**, the resizing operation is cancelled on bothCFs. The size of the structure on the secondary CF is then adjusted to match the value that is on the primary CF. You can monitor a resize request by running the command `DB2 GET DB CFG SHOW DETAIL`.

Ingest utility configuration parameters

commit_count - Commit count configuration parameter

This parameter specifies the number of rows each flusher writes in a single transaction before issuing a commit.

Configuration type
Ingest utility

Applies to
Ingest utility

Parameter type
Configurable

Default [range]
0 [0 to max 32-bit signed integer]

If **commit_count** is not a multiple of 1,000, it is rounded to the nearest multiple and the ingest utility issues a warning (SQL2903W).

When **commit_count** is set to 0 (the default), **commit_period** is used, meaning that by default, the ingest utility commits transactions based on elapsed time only. If you want to commit transactions based on the number of rows only, you must set **commit_count** to a non-zero value and set **commit_period** to 0.

When neither **commit_count** nor **commit_period** is specified, the **commit_period** default setting of 1 second is used.

If both **commit_count** and **commit_period** are specified, the ingest utility honors both; that is, it issues a commit when it has written the specified number of rows or if there has not been a commit within the specified number of seconds.

Recommendations

If the row size is small, set **commit_count** to a large value. If the row size is large, set **commit_count** to a small value.

If no other applications are running, the absolute maximum commit count can be estimated very roughly using the following formula:

```
(logfilsiz * (logprimary + logsecond) * 4KB) divided by (estimated row size + overhead)
   divided by (total number of flushers)
```

If other applications are running, the maximum commit count is smaller.

If **commit_count** is set to too large a value, it is likely the lock list or the transaction log will fill up. In that case, the INGEST command terminates with error SQL0912N or SQL0964C. If you get SQL0912N or SQL0964C and you have the **retry_count** configuration parameter set, the ingest utility does both of the following

- adjusts the setting of either **commit_count** or **commit_period**, or both and issues a warning message
- issues a commit and retries the operation

commit_period - Commit period configuration parameter

Specifies the number of seconds between committed transactions.

Configuration type
Ingest utility

Applies to
Ingest utility

Parameter type
Configurable

Default [range]
1 [0 to 2,678,400 (31 days)]

Unit of measure
Seconds

At each flush, the ingest utility checks the number of seconds since the last commit. If it is greater than or equal to the setting of **commit_period**, the ingest utility issues a commit.

If **commit_period** is set to 0, the **commit_count** configuration parameter is used, meaning that transactions are to be committed based upon the number of rows.

When neither **commit_count** nor **commit_period** is specified, the **commit_period** default setting of 1 second is used.

If both **commit_count** and **commit_period** are specified, the ingest utility honors both; that is, it issues a commit when it has written the specified number of rows or if there has not been a commit within the specified number of seconds.

Recommendations
If the row size is small, set **commit_period** to a large value. If the row size is large, set **commit_period** to a small value.

If **commit_period** is set to too large a value, it is likely the lock list or the transaction log will fill up. In that case, the INGEST command terminates with error SQL0912N or SQL0964C. If you get SQL0912N or SQL0964C and you have the **retry_count** configuration parameter set, the ingest utility does both of the following actions

- adjusts the setting of either **commit_count** or **commit_period**, or both and issues a warning message
- issues a commit and retries the operation

num_flushers_per_partition - Number of flushers per database partition configuration parameter

Specifies the number of flushers to allocate for each database partition.

Configuration type
Ingest utility

Applies to
Ingest utility

Parameter type
Configurable

Default [range]
max(1, ((number of logical CPUs)/2)/(number of partitions)) [0 to system resources]

When **num_flushers_per_partition** is set to 0, one flusher is used for all partitions.

Note: When the operation is DELETE, MERGE, or UPDATE, the utility will sometimes reduce this parameter to 0 or 1 in order to reduce the chances of a deadlock occurring (SQL2903W).

num_formatters - Number of formatters configuration parameter

Specifies the number of formatters to allocate.

Configuration type
Ingest utility

Applies to
Ingest utility

Parameter type
Configurable

Default [range]
max(1, ((number of logical CPUs)/2)) [1 to system resources]

Although the default is the optimal setting, if you specify the `DUMPFILE` (or `BADFILE`) parameter on the **INGEST** command and you want the ingest utility to write the bad records in the same order that they appear in the input file, you must set **num_formatters** to 1.

pipe_timeout - Pipe timeout configuration parameter

This parameter specifies the maximum number of seconds to wait for data when the input source is a pipe.

Configuration type
Ingest command

Applies to
Ingest utility

Parameter type
Configurable

Default [range]
600 seconds (10 minutes) [0 to 2,678,400 (31 days)]

Unit of measure
seconds

When **pipe_timeout** is set to 0, the ingest utility waits indefinitely. If no data arrives within the specified period, the **INGEST** command ends and an error is returned.

retry_count - Retry count configuration parameter

Specifies the number of times to retry a failed, but recoverable, transaction.

Configuration type
Ingest utility

Applies to
Ingest utility

Parameter type
Configurable

Default [range]
0 [0 to 1,000]

The ingest utility only retries transactions that fail for one of the following reasons:

- A connection failed but has been reestablished.
- Deadlock or timeout with automatic rollback occurred.
- A system error has caused the unit of work to be rolled back.
- Virtual storage or database resource is not available.

retry_period - Retry period configuration parameter

Specifies the number of seconds to wait before retrying a failed, but recoverable, transaction.

Configuration type
Ingest utility

Applies to
Ingest utility

Parameter type
Configurable

Default [range]
0 [0 to 2,678,400 (31 days)]

Unit of measure
Seconds

The ingest utility only retries transactions that fail for one of the following reasons:

- A connection failed but has been reestablished.
- Deadlock or timeout with automatic rollback occurred.
- A system error has caused the unit of work to be rolled back.
- Virtual storage or database resource is not available.

shm_max_size - Maximum size of shared memory configuration parameter

Specifies the maximum size of Inter Process Communication (IPC) shared memory in bytes. Because the ingest utility runs on the client, this memory is allocated on the client machine.

Configuration type
Ingest command

Applies to
Ingest utility

Parameter type
Configurable

Default [range]
1 GB (1,073,741,824) [*n* to available memory]

n is calculated as follows:

```
11000 +
(numTransporters x 500) +
(NUM_FORMATTERS x 500) +
(numUsedPartitions x 50) +
(totalNumFlushers x 4000) +
(MSG_BUF_COUNT x (100 + MSG_BUF_SIZE)) +
(numFields x 66300) +
(1.5 x NUM_FORMATTERS x sumOfAllFieldLengths)
```

where

- *numTransporters* is the number of input sources (if operation is INSERT or REPLACE), or 1 otherwise.
- *numUsedPartitions* is number of database partitions that the target table uses.
- *totalNumFlushers* is **num_flushers_per_partition** x *numUsedPartitions*.
- *sumOfAllFieldLengths* is the total number of bytes in all the field definitions.
- *numFields* is the number of field definitions.

Unit of measure
bytes

Part 5. 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 140. 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 140. 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	SC27-4541-00	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 141. 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 141. 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.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM 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 www.ibm.com/legal/copytrade.shtml

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.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Printed in USA

SC27-3871-01

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

Database Administration Concepts and Configuration Reference