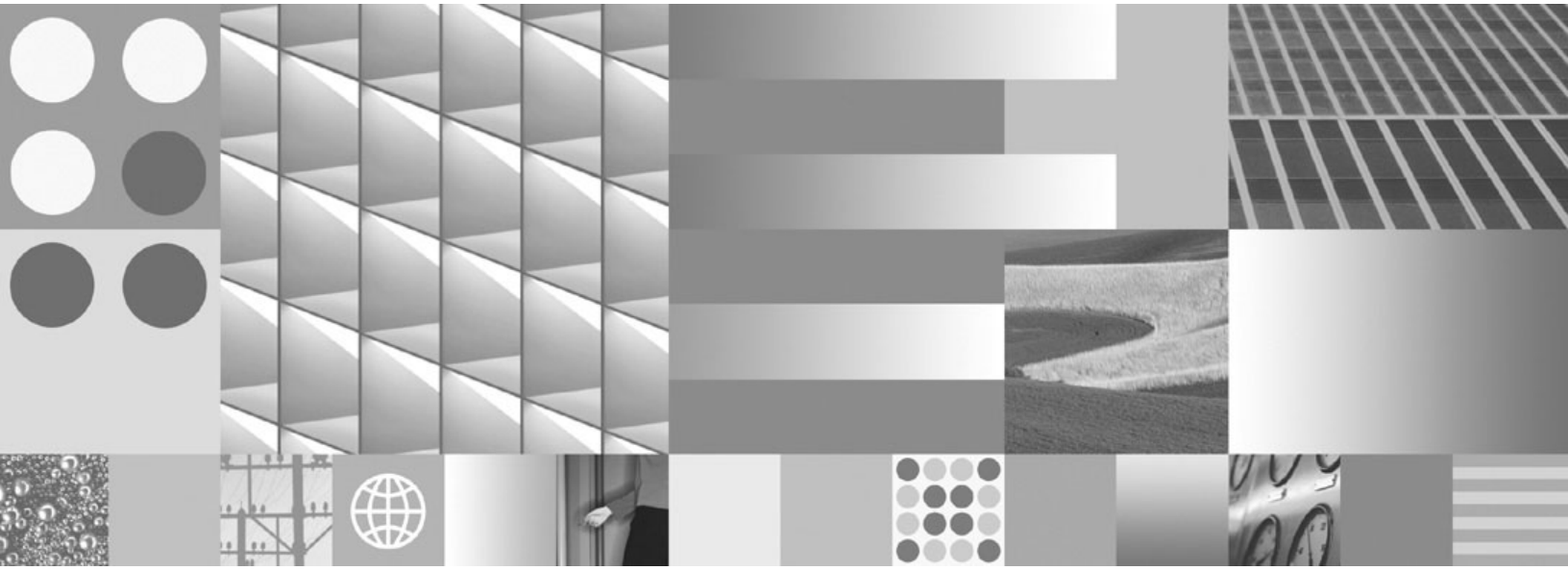
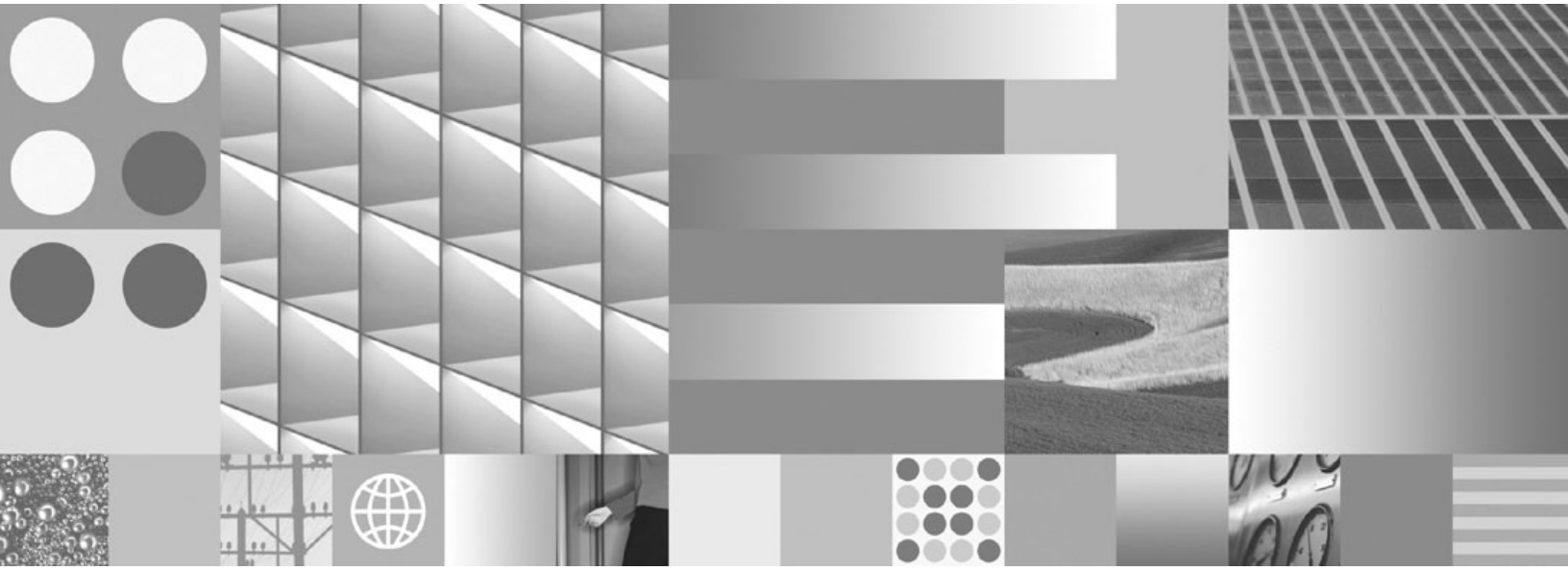
IBM DB2 9.7
for Linux, UNIX, and Windows



Common Criteria Certification: Administration and User Documentation - Volume 1 - Revision 6

SC14-7213-00

IBM DB2 9.7
for Linux, UNIX, and Windows



Common Criteria Certification: Administration and User Documentation - Volume 1 - Revision 6

SC14-7213-00

Note
Before using this information and the product it supports, read the general information under Appendix C, "Notices," on page 1097.

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

Common Criteria certification of DB2 products

For Version 9.7, IBM® DB2® products are certified according to the Common Criteria evaluation assurance level 4 (EAL4), augmented with Flaw remediation ALC_FLR.1.

The following product is certified on the following operating systems:

Table 1.

	Windows® Server 2003	**Red Hat Enterprise Linux® 5**	**SuSE Linux Enterprise Server 10**	**AIX® 6**	**Solaris 10**
IBM DB2 Version 9.7 Enterprise Server Edition for Linux, UNIX®, and Windows	Yes	Yes	Yes	Yes	Yes

Note:

1. For a Common Criteria certified DB2 environment, DB2 requires 64-bit Windows Server 2003 x64, Red Hat Enterprise Linux 5, or SuSE Linux Enterprise Server 10 operating systems for Intel® EM64T- and AMD64-based systems.
2. In a Common Criteria certified DB2 environment, DB2 clients are supported on the following operating systems:
   - Windows 2003
   - Red Hat Enterprise Linux 5
   - SuSE Linux Enterprise Server 10
   - AIX 6
   - Solaris 10

For more information about Common Criteria, see the Common Criteria web site at: `http://www.commoncriteriaportal.org`.

For information about installing and configuring a DB2 system that conforms to the Common Criteria EAL4, see the following books:

- *Installing IBM DB2 Enterprise Server Edition*
- *IBM DB2 Administration and User Documentation*

These books are available in PDF format from the *DB2 Information Management Library*.

Supported interfaces for a Common Criteria evaluated configuration

The set of DB2 interfaces that are used in the Common Criteria evaluation of the DB2 database manager are as follows:

- The DB2 install program
- The command line processor
- DB2 commands
- DB2 application programming interfaces (APIs)
- SQL statements

You can use these interfaces when installing and configuring a Common Criteria compliant DB2 system.

Other interfaces that are provided by the DB2 database manager, such as the Control Center or Command Editor were not used during the Common Criteria evaluation of DB2 products, **and must not be used in the Common Criteria evaluation configuration.**

Note:

- NOT FENCED routines are also not supported.
- Data encryption functions ENCRYPT, DECRYPT_BIN, DECRYPT_CHAR and GETHINT must not be used.
- The user-written security plugins must not be used.
- Like table privileges (SELECT, INSERT, UPDATE, DELETE), Label-Based Access Control (LBAC) has no control over access to physical files such as database files and transaction logs. Given that these files contain database data including data protected with LBAC and given that DB2 administrators have direct access to these physical files, DB2 administrators should be treated as having the highest level of access even though this is an indirect access and is outside the scope of the LBAC model.

When using the DB2 Database Partition Feature (DPF), the external security information that is used by the DB2 database manager to perform authentication and authorization must be configured consistently on each partition. This information depends on the authentication type in use. For operating system security, this information is the username, password and group membership of each user that can connect to the database. Identical usernames, passwords and groups must be created at each partition. For LDAP authentication, this information is stored in the LDAP configuration file for the LDAP-based authentication plugin. The LDAP configuration file must have the same contents on each partition. For Kerberos, in order to ensure that the same Key Distribution Center (KDC) is used for each partition, this information is stored in the Kerberos configuration file on the server where the DB2 product is installed. Failure to provide consistent configuration information at each partition could result in users being unable to authenticate, and hence connect to the DB2 database, or users having reduced privileges, if incomplete group membership information is obtained from the local operating system, or from LDAP, or from Kerberos.

About this book

This book, consisting of volumes 1 and 2, is intended for use by assessors validating that specific DB2 database products conform to the Common Criteria EAL4 specification augmented with Flaw remediation ALC_FLR.1. It is also intended for those who want to set up a DB2 environment that conforms to the characteristics of the evaluated environment.

Volume 1 describes:

- The DB2 process model
- The DB2 security model, and the facilities available to set up and maintain security
- How to set up the DB2 environment so that it conforms to the requirements of the Common Criteria EAL4 specification
- How to audit activity in the environment
- Background information that you should be familiar with before setting up the DB2 database environment
- Security-related considerations that are applicable to users of the DB2 database environment, including the type of authorization that the administrator must give to a user before that user can work with DB2 utilities.
- DB2 commands.

Regarding security considerations on SQL statements and SQL routines (found in chapters 5 and 6):

- Passwords appear in SQL statements in plain text. As such, any program or script containing such statements needs appropriate protection with OS- and DBMS-provided mechanisms.
- A major database vulnerability (generic) is SQL injection. As such, use caution and validate any direct user input looking for SQL injection attacks—looking for SQL statements, special characters such as {},; and quotes.

Note: This book does not provide information on how to install DB2 database servers. For installation information, See the Version 9.7 *Installing IBM DB2 Enterprise Server Edition*.

Some topics in book link to related topics, which are either included in Appendix A in order to resolve the links, or that are referenced outside of the Common Criteria certification documentation. These are for informational purposes only, and are not required for either installing or configuring a Common Criteria compliant environment.

Part 1. Overview of the DB2 environment

Chapter 1. DB2 architecture and process overview

On the client side, local or remote applications are linked with the DB2 client library. Local clients communicate using shared memory and semaphores; remote clients use a protocol, such as named pipes (NPIPE) or TCP/IP. On the server side, activity is controlled by engine dispatchable units (EDUs).

Figure 1 shows a general overview of the DB2 architecture and processes.

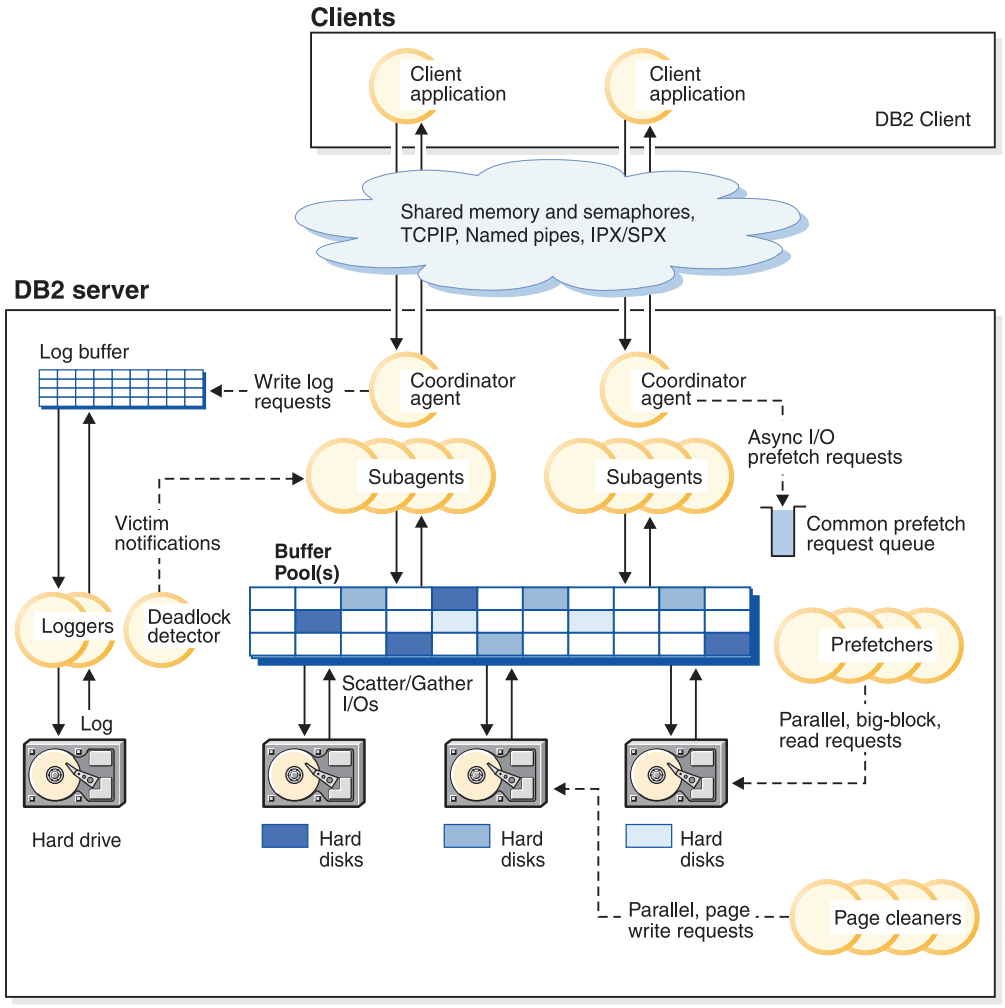


Figure 1. Client connections and database server components

EDUs are shown as circles or groups of circles.

EDUs are implemented as threads on all platforms. DB2 agents are the most common type of EDU. These agents perform most of the SQL and XQuery processing on behalf of applications. Prefetchers and page cleaners are other common EDUs.

A set of subagents might be assigned to process client application requests. Multiple subagents can be assigned if the machine on which the server resides has

multiple processors or is part of a partitioned database environment. For example, in a symmetric multiprocessing (SMP) environment, multiple SMP subagents can exploit multiple processors.

All agents and subagents are managed by a pooling algorithm that minimizes the creation and destruction of EDUs.

Buffer pools are areas of database server memory where pages of user data, index data, and catalog data are temporarily moved and can be modified. Buffer pools are a key determinant of database performance, because data can be accessed much faster from memory than from disk.

The configuration of buffer pools, as well as prefetcher and page cleaner EDUs, controls how quickly data can be accessed by applications.

- *Prefetchers* retrieve data from disk and move it into a buffer pool before applications need the data. For example, applications that need to scan through large volumes of data would have to wait for data to be moved from disk into a buffer pool if there were no data prefetchers. Agents of the application send asynchronous read-ahead requests to a common prefetch queue. As prefetchers become available, they implement those requests by using big-block or scatter-read input operations to bring the requested pages from disk into the buffer pool. If you have multiple disks for data storage, the data can be striped across those disks. Striping enables the prefetchers to use multiple disks to retrieve data simultaneously.
- *Page cleaners* move data from a buffer pool back to disk. Page cleaners are background EDUs that are independent of the application agents. They look for pages that have been modified, and write those changed pages out to disk. Page cleaners ensure that there is room in the buffer pool for pages that are being retrieved by prefetchers.

Without the independent prefetchers and page cleaner EDUs, the application agents would have to do all of the reading and writing of data between a buffer pool and disk storage.

Chapter 2. The DB2 process model

Knowledge of the DB2 process model will help you to understand how the database manager and its associated components interact, and this can help you to troubleshoot problems that might arise.

The process model that is used by all DB2 database servers facilitates communication between database servers and clients. It also ensures that database applications are isolated from resources, such as database control blocks and critical database files.

The DB2 database server must perform many different tasks, such as processing database application requests or ensuring that log records are written out to disk. Each task is typically performed by a separate *engine dispatchable unit* (EDU).

There are many advantages to using a multithreaded architecture for the DB2 database server. A new thread requires less memory and fewer operating system resources than a process, because some operating system resources can be shared among all threads within the same process. Moreover, on some platforms, the context switch time for threads is less than that for processes, which can improve performance. Using a threaded model on all platforms makes the DB2 database server easier to configure, because it is simpler to allocate more EDUs when needed, and it is possible to dynamically allocate memory that must be shared by multiple EDUs.

For each database being accessed, separate EDUs are started to deal with various database tasks such as prefetching, communication, and logging. Database agents are a special class of EDU that are created to handle application requests for a database.

In general, you can rely on the DB2 database server to manage the set of EDUs. However, there are DB2 tools that look at the EDUs. For example, you can use the db2pd command with the **-edus** option to list all EDU threads that are active.

Each client application connection has a single coordinator agent that operates on a database. A *coordinator agent* works on behalf of an application, and communicates to other agents using private memory, interprocess communication (IPC), or remote communication protocols, as needed.

The DB2 architecture provides a firewall so that applications run in a different address space than the DB2 database server (Figure 2 on page 6). The firewall protects the database and the database manager from applications, stored procedures, and user-defined functions (UDFs). The firewall maintains the integrity of the data in the databases, because it prevents application programming errors from overwriting internal buffers or database manager files. The firewall also improves reliability, because application errors cannot crash the database manager.

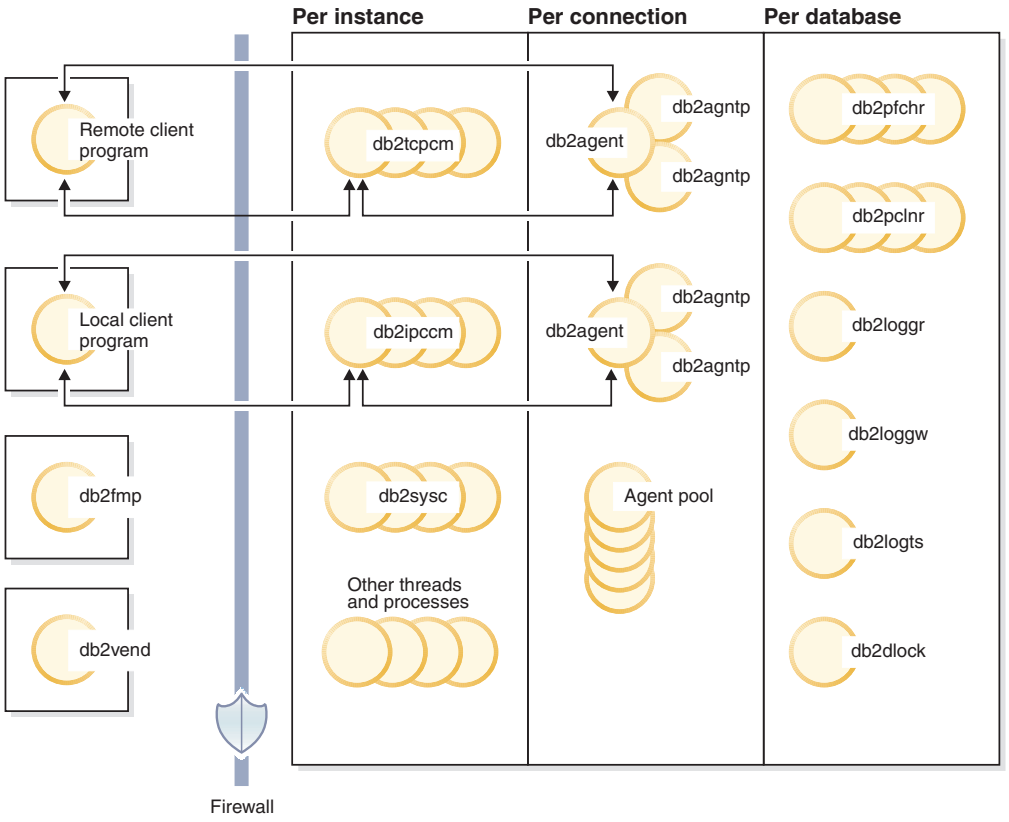


Figure 2. Process model for DB2 database systems

Client programs

Client programs can be remote or local, running on the same machine as the database server. Client programs make first contact with a database through a communication listener.

Listeners

Communication listeners start when the DB2 database server starts. There is a listener for each configured communications protocol, and an interprocess communications (IPC) listener (db2ipccm) for local client programs. Listeners include:

- db2ipccm, for local client connections
- db2tcpcm, for TCP/IP connections
- db2tcpdm, for TCP/IP discovery tool requests

Agents

All connection requests from local or remote client programs (applications) are allocated a corresponding coordinator agent (db2agent). When the coordinator agent is created, it performs all database requests on behalf of the application.

In partitioned database environments, or systems on which *intraquery parallelism* has been enabled, the coordinator agent distributes database requests to subagents (db2agntp and db2agnts, respectively). Subagents that are associated with an application but that are currently idle are named db2agnta.

A coordinator agent might be:

- Connected to the database with an alias; for example, db2agent (DATA1) is connected to the database alias DATA1.
- Attached to an instance; for example, db2agent (user1) is attached to the instance user1.

The DB2 database server will instantiate other types of agents, such as independent coordinator agents or subcoordinator agents, to execute specific operations. For example, the independent coordinator agent db2agnti is used to run event monitors, and the subcoordinator agent db2agnsc is used to parallelize database restart operations following an abnormal shutdown.

Idle agents reside in an agent pool. These agents are available for requests from coordinator agents operating on behalf of client programs, or from subagents operating on behalf of existing coordinator agents. Having an appropriately-sized idle agent pool can improve performance when there are significant application workloads. In this case, idle agents can be used as soon as they are required, and there is no need to allocate a completely new agent for each application connection, which involves creating a thread and allocating and initializing memory and other resources. The DB2 database server automatically manages the size of the idle agent pool.

db2fmp

The fenced mode process is responsible for executing fenced stored procedures and user-defined functions outside of the firewall. The db2fmp process is always a separate process, but might be multithreaded, depending on the types of routines that it executes.

db2vend

This is a process to execute vendor code on behalf of an EDU; for example, to execute a user exit program for log archiving (UNIX only).

Database EDUs

The following list includes some of the important EDUs that are used by each database:

- db2pfchr, for buffer pool prefetchers
- db2pclnr, for buffer pool page cleaners
- db2loggr, for manipulating log files to handle transaction processing and recovery
- db2loggw, for writing log records to the log files
- db2logts, for tracking which table spaces have log records in which log files. This information is recorded in the `DB2TSCHG.HIS` file in the database directory.
- db2dlock, for deadlock detection. In a partitioned database environment, an additional thread (db2glock) is used to coordinate the information that is collected by the db2dlock EDU on each partition; db2glock runs only on the catalog partition.
- db2stmm, for the self-tuning memory management feature
- db2taskd, for the distribution of background database tasks. These tasks are executed by threads called db2taskp.
- db2hadrp, the high availability disaster recovery (HADR) primary server thread

- db2hadrs, the HADR standby server thread
- db2lfr, for log file readers that process individual log files
- db2shred, for processing individual log records within log pages
- db2redom, for the redo master. During recovery, it processes redo log records and assigns log records to redo workers for processing.
- db2redow, for the redo workers. During recovery, it processes redo log records at the request of the redo master.
- db2logmgr, for the log manager. Manages log files for a recoverable database.
- db2wlmd, for automatic collection of workload management statistics
- Event monitor threads are identified as follows:
  - db2evm*%1%2* (*%3*)

    where *%1* can be:
    - g - global file event monitor
    - gp - global piped event monitor
    - l - local file event monitor
    - lp - local piped event monitor
    - t - table event monitor

    and *%2* can be:
    - i - coordinator
    - p - not coordinator

    and *%3* is the event monitor name
- Backup and restore threads are identified as follows:
  - db2bm.*%1.%2* (backup and restore buffer manipulator) and db2med.*%1.%2* (backup and restore media controller), where:
    - *%1* is the EDU ID of the agent that controls the backup or restore session
    - *%2* is a sequential value that is used to distinguish among (possibly many) threads belonging to a particular backup or restore session

    For example: **db2bm.13579.2** identifies the second db2bm thread that is controlled by the db2agent thread with EDU ID 13579.

Database server threads and processes

The system controller (db2sysc on UNIX and db2syscs.exe on Windows operating systems) must exist if the database server is to function. The following threads and processes carry out a variety of tasks:

- db2resync, the resync agent that scans the global resync list
- db2wdog, the watchdog on UNIX and Linux operating systems that handles abnormal terminations
- db2fcms, the fast communications manager sender daemon
- db2fcmr, the fast communications manager receiver daemon
- db2pdbc, the parallel system controller, which handles parallel requests from remote database partitions (used only in a partitioned database environment)
- db2cart, for archiving log files (when the **userexit** database configuration parameter is enabled)
- db2fmtlg, for formatting log files (when the **logretain** database configuration parameter is enabled and the **userexit** database configuration parameter is disabled)

- db2panic, the panic agent, which handles urgent requests after agent limits have been reached at a particular database partition (used only in a partitioned database environment)
- db2srvlst, manages lists of addresses for systems such as DB2 for z/OS®
- db2fmd, the fault monitor daemon
- db2disp, the client connection concentrator dispatcher
- db2acd, an autonomic computing daemon that hosts the health monitor, automatic maintenance utilities, and the administrative task scheduler. This process was formerly known as db2hmon.
- db2licc, manages installed DB2 licenses
- db2thcln, recycles resources when an EDU terminates (UNIX only)
- db2aiothr, manages asynchronous I/O requests for a database partition (UNIX only)
- db2alarm, notifies EDUs when their requested timer has expired (UNIX only)
- db2sysc, the main system controller EDU; it handles critical DB2 server events

Chapter 3. Client-server processing model

Both local and remote application processes can work with the same database. A remote application is one that initiates a database action from a machine that is remote from the machine on which the database server resides. Local applications are directly attached to the database at the server machine.

How client connections are managed depends on whether the connection concentrator is on or off. The connection concentrator is on whenever the value of the **max_connections** database manager configuration parameter is larger than the value of the **max_coordagents** configuration parameter.

- If the connection concentrator is off, each client application is assigned a unique engine dispatchable unit (EDU) called a *coordinator agent* that coordinates the processing for that application and communicates with it.
- If the connection concentrator is on, each coordinator agent can manage many client connections, one at a time, and might coordinate the other worker agents to do this work. For internet applications with many relatively transient connections, or applications with many relatively small transactions, the connection concentrator improves performance by allowing many more client applications to be connected concurrently. It also reduces system resource use for each connection.

In Figure 3 on page 12, each circle in the DB2 server represents an EDU that is implemented using operating system threads.

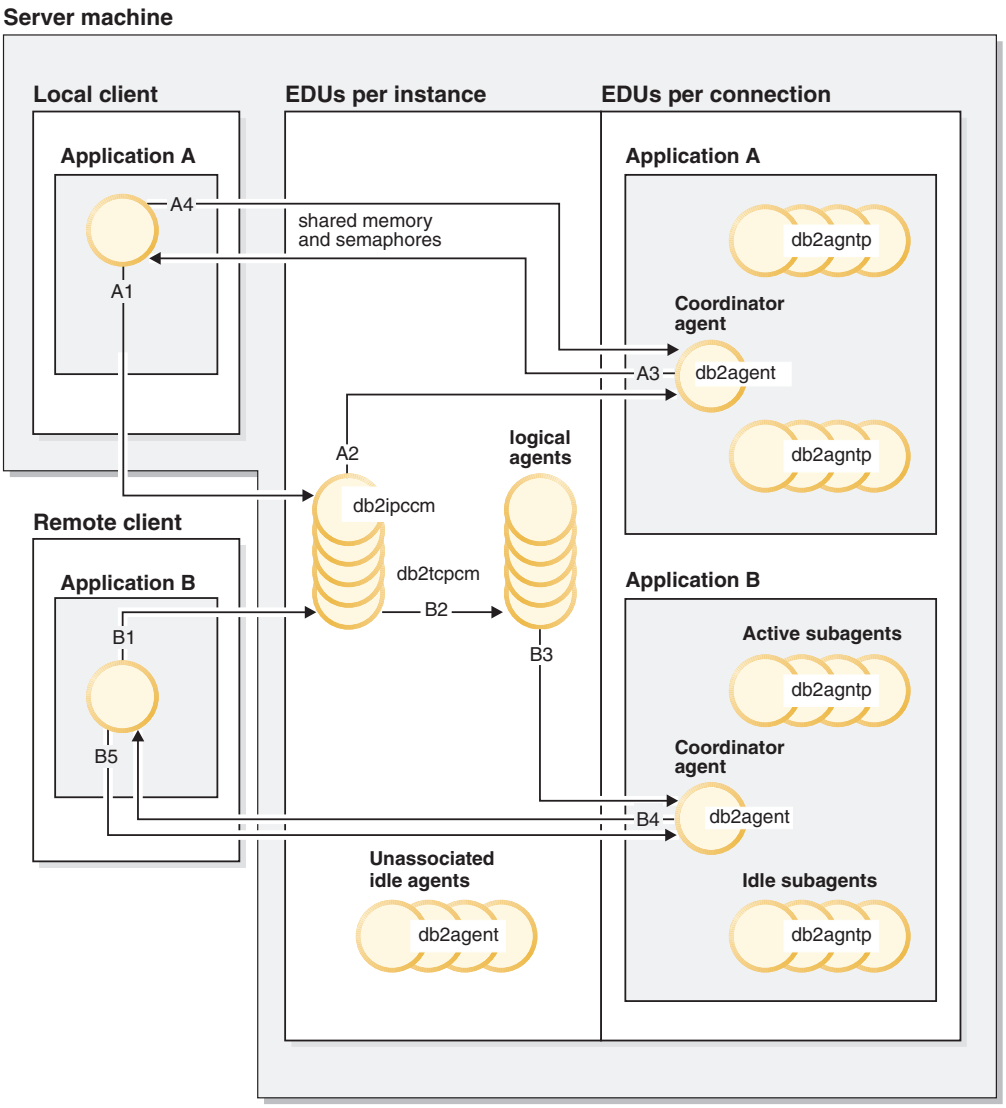


Figure 3. Client-server processing model overview

- At A1, a local client establishes communications through db2ipccm.
- At A2, db2ipccm works with a db2agent EDU, which becomes the coordinator agent for application requests from the local client.
- At A3, the coordinator agent contacts the client application to establish shared memory communications between the client application and the coordinator.
- At A4, the application at the local client connects to the database.
- At B1, a remote client establishes communications through db2tcpcm. If another communications protocol was chosen, the appropriate communications manager is used.
- At B2, db2tcpcm works with a db2agent EDU, which becomes the coordinator agent for the application and passes the connection to this agent.
- At B4, the coordinator agent contacts the remote client application.
- At B5, the remote client application connects to the database.

Note also that:

- Worker agents carry out application requests. There are four types of worker agents: active coordinator agents, active subagents, associated subagents, and idle agents.
- Each client connection is linked to an active coordinator agent.
- In a partitioned database environment, or an environment in which intra-partition parallelism is enabled, the coordinator agents distribute database requests to subagents (db2agntp).
- There is an agent pool (db2agent) where idle agents wait for new work.
- Other EDUs manage client connections, logs, two-phase commit operations, backup and restore operations, and other tasks.

Figure 4 shows additional EDUs that are part of the server machine environment. Each active database has its own shared pool of prefetchers (db2pfchr) and page cleaners (db2pclnr), and its own logger (db2loggr) and deadlock detector (db2dlock).

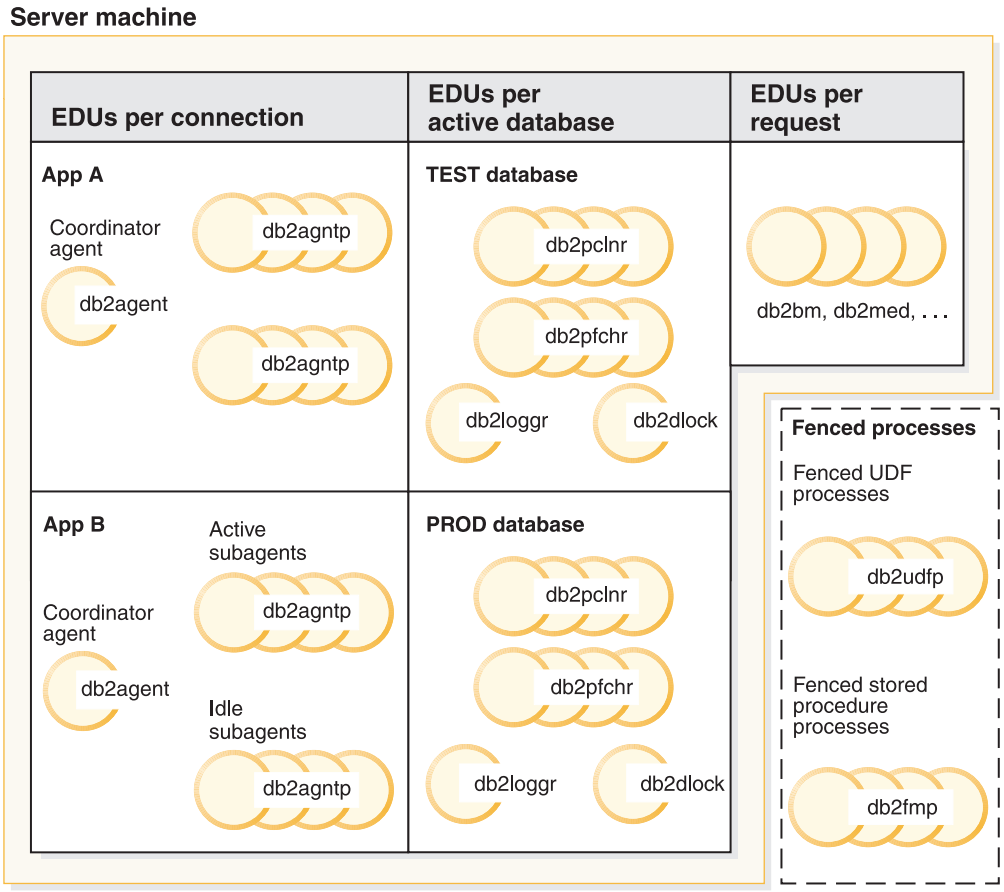


Figure 4. EDUs in the database server

Fenced user-defined functions (UDFs) and stored procedures, which are not shown in the figure, are managed to minimize costs that are associated with their creation and destruction. The default value of the **keepfenced** database manager configuration parameter is YES, which keeps the stored procedure process available for reuse at the next procedure call.

Note: Unfenced UDFs and stored procedures run directly in an agent's address space for better performance. However, because they have unrestricted access to the agent's address space, they must be rigorously tested before being used.

Figure 5 shows the similarities and differences between the single database partition processing model and the multiple database partition processing model.

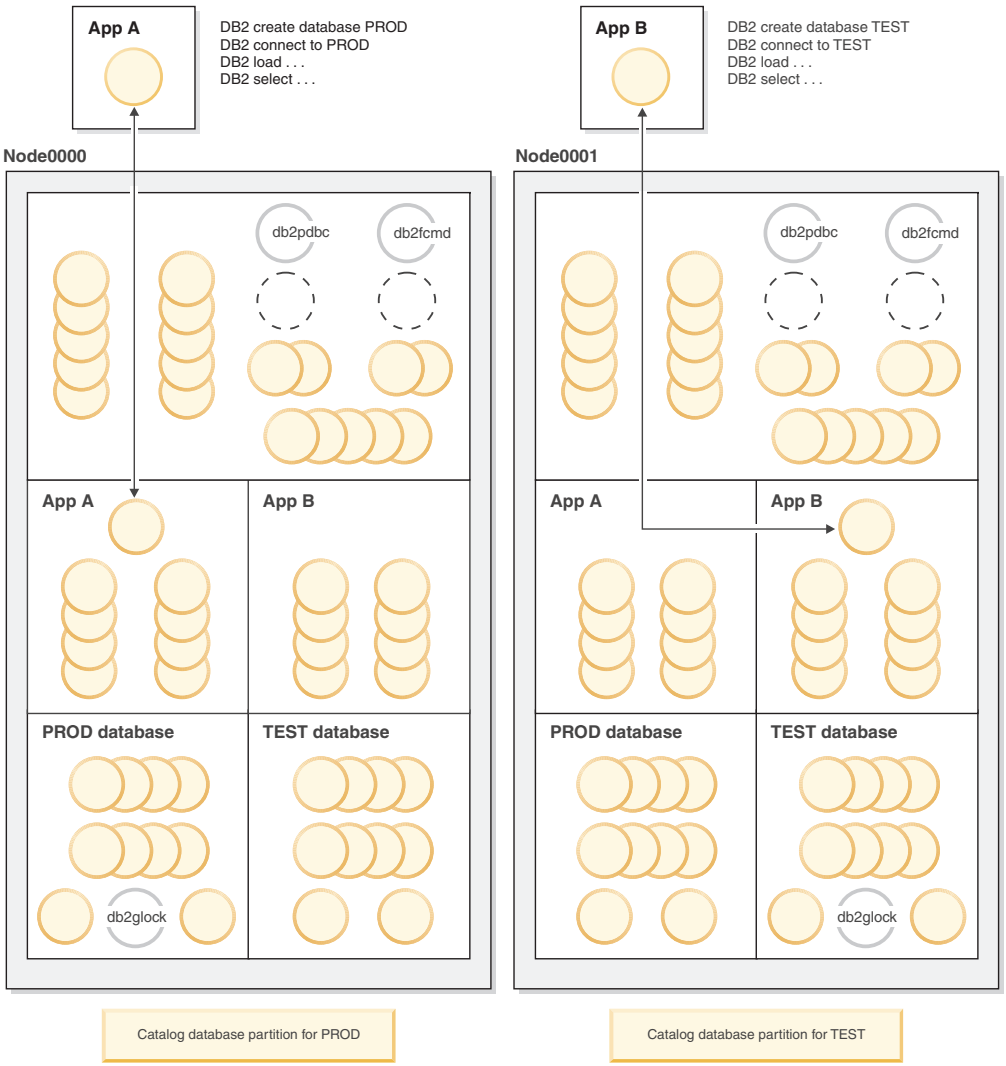


Figure 5. Process model for multiple database partitions

In a multiple database partition environment, the database partition on which the CREATE DATABASE command was issued is called the *catalog database partition*. It is on this database partition that the system catalog tables are stored. The system catalog is a repository of all of the information about objects in the database.

As shown in Figure 5, because Application A creates the PROD database on Node0000, the catalog for the PROD database is also created on this database partition. Similarly, because Application B creates the TEST database on Node0001, the catalog for the TEST database is created on this database partition. It is a good idea to create your databases on different database partitions to balance the extra activity that is associated with the catalog for each database across the database partitions in your environment.

There are additional EDUs (db2pdbc and db2fcmd) that are associated with the instance, and these are found on each database partition in a multiple database partition environment. These EDUs are needed to coordinate requests across database partitions and to enable the fast communication manager (FCM).

There is an additional EDU (db2glock) that is associated with the catalog database partition. This EDU controls global deadlocks across the database partitions on which the active database is located.

Each connect request from an application is represented by a connection that is associated with a coordinator agent. The *coordinator agent* is the agent that communicates with the application, receiving requests and sending replies. It can satisfy a request itself or coordinate multiple subagents to work on the request. The database partition on which the coordinator agent resides is called the *coordinator database partition* of that application.

Parts of the database requests from an application are sent by the coordinator database partition to subagents at the other database partitions. All of the results are consolidated at the coordinator database partition before being sent back to the application.

Any number of database partitions can be configured to run on the same machine. This is known as a *multiple logical partition* configuration. Such a configuration is very useful on large symmetric multiprocessor (SMP) machines with very large main memory. In this environment, communications between database partitions can be optimized to use shared memory and semaphores.

Chapter 4. Database agents

When an application accesses a database, several processes or threads begin to perform the various application tasks. These tasks include logging, communication, and prefetching. Database agents are threads within the database manager that are used to service application requests. In Version 9.5, agents are run as threads on all platforms.

The maximum number of application connections is controlled by the **max_connections** database manager configuration parameter. The work of each application connection is coordinated by a single worker agent. A *worker agent* carries out application requests but has no permanent attachment to any particular application. *Coordinator agents* exhibit the longest association with an application, because they remain attached to it until the application disconnects. The only exception to this rule occurs when the engine concentrator is enabled, in which case a coordinator agent can terminate that association at transaction boundaries (COMMIT or ROLLBACK).

There are three types of worker agents:

- Idle agents

  This is the simplest form of worker agent. It does not have an outbound connection, and it does not have a local database connection or an instance attachment.
- Active coordinator agents

  Each database connection from a client application has a single active agent that coordinates its work on the database. After the coordinator agent is created, it performs all database requests on behalf of its application, and communicates to other agents using interprocess communication (IPC) or remote communication protocols. Each agent operates with its own private memory and shares database manager and database global resources, such as the buffer pool, with other agents. When a transaction completes, the active coordinator agent might become an inactive agent. When a client disconnects from a database or detaches from an instance, its coordinator agent will be:
  - An active coordinator agent if other connections are waiting
  - Freed and marked as idle if no connections are waiting, and the maximum number of pool agents is being automatically managed or has not been reached
  - Terminated and its storage freed if no connections are waiting, and the maximum number of pool agents has been reached
- Subagents

  The coordinator agent distributes database requests to subagents, and these subagents perform the requests for the application. After the coordinator agent is created, it handles all database requests on behalf of its application by coordinating the subagents that perform requests against the database. In DB2 Version 9.5, subagents can also exist in nonpartitioned environments and in environments where intraquery parallelism is not enabled.

Agents that are not performing work for any application and that are waiting to be assigned are considered to be idle agents and reside in an *agent pool*. These agents are available for requests from coordinator agents operating on behalf of client

programs, or for subagents operating on behalf of existing coordinator agents. The number of available agents depends on the value of the **num_poolagents** database manager configuration parameter.

If no idle agents exist when an agent is required, a new agent is created dynamically. Because creating a new agent requires a certain amount of overhead, CONNECT and ATTACH performance is better if an idle agent can be activated for a client.

When a subagent is performing work for an application, it is associated with that application. After it completes the assigned work, it can be placed in the agent pool, but it remains associated with the original application. When the application requests additional work, the database manager first checks the idle pool for associated agents before it creates a new agent.

Chapter 5. Database agent management

Most applications establish a one-to-one relationship between the number of connected applications and the number of application requests that can be processed by the database server. Your environment, however, might require a many-to-one relationship between the number of connected applications and the number of application requests that can be processed.

Two database manager configuration parameters control these factors separately:

- The **max_connections** parameter specifies the maximum number of connected applications
- The **max_coordagents** parameter specifies the maximum number of application requests that can be processed concurrently

The connection concentrator is enabled when the value of **max_connections** is greater than the value of **max_coordagents**. Because each active coordinating agent requires global database resource overhead, the greater the number of these agents, the greater the chance that the upper limits of available global resources will be reached. To prevent this from occurring, set the value of **max_connections** to be higher than the value of **max_coordagents**, or set both parameters to AUTOMATIC.

There are two specific scenarios in which setting these parameters to AUTOMATIC is a good idea:

- If you are confident that your system can handle all of the connections that might be needed, but you want to limit the amount of global resources that are used (by limiting the number of coordinating agents), set **max_connections** to AUTOMATIC. When **max_connections** is greater than **max_coordagents**, the connection concentrator is enabled.
- If you do not want to limit the maximum number of connections or coordinating agents, but you know that your system requires or would benefit from a many-to-one relationship between the number of connected applications and the number of application requests that are processed, set both parameters to AUTOMATIC. When both parameters are set to AUTOMATIC, the database manager uses the values that you specify as an ideal ratio of connections to coordinating agents. Note that both of these parameters can be configured with a starting value and an AUTOMATIC setting. For example, the following command associates both a value of 200 and AUTOMATIC with the **max_coordagents** parameter:`update dbm config using max_coordagents 200 automatic`.

Example

Consider the following scenario:

- The **max_connections** parameter is set to AUTOMATIC and has a current value of 300
- The **max_coordagents** parameter is set to AUTOMATIC and has a current value of 100

The ratio of **max_connections** to **max_coordagents** is 300:100. The database manager creates new coordinating agents as connections come in, and connection concentration is applied only when needed. These settings result in the following actions:

- Connections 1 to 100 create new coordinating agents
- Connections 101 to 300 do not create new coordinating agents; they share the 100 agents that have been created already
- Connections 301 to 400 create new coordinating agents
- Connections 401 to 600 do not create new coordinating agents; they share the 200 agents that have been created already
- and so on...

In this example, it is assumed that the connected applications are driving enough work to warrant creation of new coordinating agents at each step. After some period of time, if the connected applications are no longer driving sufficient amounts of work, coordinating agents will become inactive and might be terminated.

If the number of connections is reduced, but the amount of work being driven by the remaining connections is high, the number of coordinating agents might not be reduced right away. The **max_connections** and **max_coordagents** parameters do not directly affect agent pooling or agent termination. Normal agent termination rules still apply, meaning that the connections to coordinating agents ratio might not correspond exactly to the values that you specified. Agents might return to the agent pool to be reused before they are terminated.

If finer granularity of control is needed, specify a simpler ratio. For example, the ratio of 300:100 from the previous example can be expressed as 3:1. If **max_connections** is set to 3 (AUTOMATIC) and **max_coordagents** is set to 1 (AUTOMATIC), one coordinating agent can be created for every three connections.

Chapter 6. Naming rules

General naming rules

Rules exist for the naming of all database objects, users, 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 quotes (")
- 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 primary 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:** Version 9.5, and later, of the DB2 database system allows you to have an 128-byte authorization ID, but when the authorization ID is interpreted as an operating system user ID or group name, the operating system naming restrictions apply (for example, Linux and UNIX operating systems have a limitation to 8 characters and Windows operating systems have a limitation of 30 characters for user IDs and group names). Therefore, while you can grant an 128-byte authorization ID, it is not possible to

connect as a user that has that authorization ID. If you write your own security plugin, you should be able to take full advantage of the extended sizes for the authorization ID. For example, you can give your security plugin a 30-byte user ID and it can return an 128-byte authorization ID during authentication that you are able to connect with.

You also must consider object naming rules, naming rules in an NLS 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 2. 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 3. 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 4. 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.

User, user ID and group naming rules

User, user ID and group names must follow naming guidelines.

Table 5. 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.

Naming rules in an NLS 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 an NLS environment because they are not included in the NLS 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.

Reserved schema names and reserved words

There are restrictions on the use of certain names that are required by the database manager. In some cases, names are reserved, and cannot be used by application programs. In other cases, certain names are not recommended for use by application programs, although their use is not prevented by the database manager.

The reserved schema names are:

- SYSCAT
- SYSFUN
- SYSIBM
- SYSIBMADM
- SYSPROC
- SYSPUBLIC
- SYSSTAT

It is strongly recommended that schema names never begin with the 'SYS' prefix, because 'SYS', by convention, is used to indicate an area that is reserved by the system. No aliases, global variables, triggers, user-defined functions, or user-defined types can be placed into a schema whose name starts with 'SYS' (SQLSTATE 42939).

The DB2QP schema and the SYSTOOLS schema are set aside for use by DB2 tools. It is recommended that users not explicitly define objects in these schemas, although their use is not prevented by the database manager.

It is also recommended that SESSION not be used as a schema name. Because declared temporary tables must be qualified by SESSION, it is possible to have an application declare a temporary table with a name that is identical to that of a persistent table, complicating the application logic. To avoid this possibility, do not use the schema SESSION except when dealing with declared temporary tables.

There are no specifically reserved words in DB2 Version 9. Keywords can be used as ordinary identifiers, except in a context where they could also be interpreted as SQL keywords. In such cases, the word must be specified as a delimited identifier. For example, COUNT cannot be used as a column name in a SELECT statement, unless it is delimited.

ISO/ANSI SQL2003 and other DB2 database products include reserved words that are not enforced by DB2 Database for Linux, UNIX, and Windows; however, it is recommended that these words not be used as ordinary identifiers, because it reduces portability.

For portability across the DB2 database products, the following should be considered reserved words:

```
ACTIVATE          DOCUMENT        LOCK              ROUND_CEILING
ADD               DOUBLE          LOCKMAX           ROUND_DOWN
AFTER             DROP            LOCKSIZE          ROUND_FLOOR
ALIAS             DSSIZE          LONG              ROUND_HALF_DOWN
ALL               DYNAMIC         LOOP              ROUND_HALF_EVEN
ALLOCATE          EACH            MAINTAINED        ROUND_HALF_UP
ALLOW             EDITPROC        MATERIALIZED      ROUND_UP
ALTER             ELSE            MAXVALUE          ROUTINE
AND               ELSEIF          MICROSECOND       ROW
```

ANY
AS
ASENSITIVE
ASSOCIATE
ASUTIME
AT
ATTRIBUTES
AUDIT
AUTHORIZATION
AUX
AUXILIARY
BEFORE
BEGIN
BETWEEN
BINARY
BUFFERPOOL
BY
CACHE
CALL
CALLED
CAPTURE
CARDINALITY
CASCADED
CASE
CAST
CCSID
CHAR
CHARACTER
CHECK
CLONE
CLOSE
CLUSTER
COLLECTION
COLLID
COLUMN
COMMENT
COMMIT
CONCAT
CONDITION
CONNECT
CONNECTION
CONSTRAINT
CONTAINS
CONTINUE
COUNT
COUNT_BIG
CREATE
CROSS
CURRENT
CURRENT_DATE
CURRENT_LC_CTYPE
CURRENT_PATH
CURRENT_SCHEMA
CURRENT_SERVER
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_TIMEZONE
CURRENT_USER
CURSOR
CYCLE
DATA
DATABASE
DATAPARTITIONNAME
DATAPARTITIONNUM
DATE
DAY
DAYS
ENABLE
ENCODING
ENCRYPTION
END
END-EXEC
ENDING
ERASE
ESCAPE
EVERY
EXCEPT
EXCEPTION
EXCLUDING
EXCLUSIVE
EXECUTE
EXISTS
EXIT
EXPLAIN
EXTERNAL
EXTRACT
FENCED
FETCH
FIELDPROC
FILE
FINAL
FOR
FOREIGN
FREE
FROM
FULL
FUNCTION
GENERAL
GENERATED
GET
GLOBAL
GO
GOTO
GRANT
GRAPHIC
GROUP
HANDLER
HASH
HASHED_VALUE
HAVING
HINT
HOLD
HOUR
HOURS
IDENTITY
IF
IMMEDIATE
IN
INCLUDING
INCLUSIVE
INCREMENT
INDEX
INDICATOR
INF
INFINITY
INHERIT
INNER
INOUT
INSENSITIVE
INSERT
INTEGRITY
INTERSECT
INTO
IS
MICROSECONDS
MINUTE
MINUTES
MINVALUE
MODE
MODIFIES
MONTH
MONTHS
NAN
NEW
NEW_TABLE
NEXTVAL
NO
NOCACHE
NOCYCLE
NODENAME
NODENUMBER
NOMAXVALUE
NOMINVALUE
NONE
NOORDER
NORMALIZED
NOT
NULL
NULLS
NUMPARTS
OBID
OF
OLD
OLD_TABLE
ON
OPEN
OPTIMIZATION
OPTIMIZE
OPTION
OR
ORDER
OUT
OUTER
OVER
OVERRIDING
PACKAGE
PADDED
PAGESIZE
PARAMETER
PART
PARTITION
PARTITIONED
PARTITIONING
PARTITIONS
PASSWORD
PATH
PIECESIZE
PLAN
POSITION
PRECISION
PREPARE
PREVVAL
PRIMARY
PRIQTY
PRIVILEGES
PROCEDURE
PROGRAM
PSID
PUBLIC
QUERY
QUERYNO
ROW_NUMBER
ROWNUMBER
ROWS
ROWSET
RRN
RUN
SAVEPOINT
SCHEMA
SCRATCHPAD
SCROLL
SEARCH
SECOND
SECONDS
SECQTY
SECURITY
SELECT
SENSITIVE
SEQUENCE
SESSION
SESSION_USER
SET
SIGNAL
SIMPLE
SNAN
SOME
SOURCE
SPECIFIC
SQL
SQLID
STACKED
STANDARD
START
STARTING
STATEMENT
STATIC
STATMENT
STAY
STOGROUP
STORES
STYLE
SUBSTRING
SUMMARY
SYNONYM
SYSFUN
SYSIBM
SYSPROC
SYSTEM
SYSTEM_USER
TABLE
TABLESPACE
THEN
TIME
TIMESTAMP
TO
TRANSACTION
TRIGGER
TRIM
TRUNCATE
TYPE
UNDO
UNION
UNIQUE
UNTIL
UPDATE
USAGE
USER
USING

DB2GENERAL	ISOBID	RANGE	VALIDPROC
DB2GENRL	ISOLATION	RANK	VALUE
DB2SQL	ITERATE	READ	VALUES
DBINFO	JAR	READS	VARIABLE
DBPARTITIONNAME	JAVA	RECOVERY	VARIANT
DBPARTITIONNUM	JOIN	REFERENCES	VCAT
DEALLOCATE	KEEP	REFERENCING	VERSION
DECLARE	KEY	REFRESH	VIEW
DEFAULT	LABEL	RELEASE	VOLATILE
DEFAULTS	LANGUAGE	RENAME	VOLUMES
DEFINITION	LATERAL	REPEAT	WHEN
DELETE	LC_CTYPE	RESET	WHENEVER
DENSE_RANK	LEAVE	RESIGNAL	WHERE
DENSERANK	LEFT	RESTART	WHILE
DESCRIBE	LIKE	RESTRICT	WITH
DESCRIPTOR	LINKTYPE	RESULT	WITHOUT
DETERMINISTIC	LOCAL	RESULT_SET_LOCATOR	WLM
DIAGNOSTICS	LOCALDATE	RETURN	WRITE
DISABLE	LOCALE	RETURNS	XMLELEMENT
DISALLOW	LOCALTIME	REVOKE	XMLEXISTS
DISCONNECT	LOCALTIMESTAMP	RIGHT	XMLNAMESPACES
DISTINCT	LOCATOR	ROLE	YEAR
DO	LOCATORS	ROLLBACK	YEARS

The following list contains the ISO/ANSI SQL2003 reserved words that are not in the previous list:

ABS	GROUPING	REGR_INTERCEPT
ARE	INT	REGR_R2
ARRAY	INTEGER	REGR_SLOPE
ASYMMETRIC	INTERSECTION	REGR_SXX
ATOMIC	INTERVAL	REGR_SXY
AVG	LARGE	REGR_SYY
BIGINT	LEADING	ROLLUP
BLOB	LN	SCOPE
BOOLEAN	LOWER	SIMILAR
BOTH	MATCH	SMALLINT
CEIL	MAX	SPECIFICTYPE
CEILING	MEMBER	SQLEXCEPTION
CHAR_LENGTH	MERGE	SQLSTATE
CHARACTER_LENGTH	METHOD	SQLWARNING
CLOB	MIN	SQRT
COALESCE	MOD	STDDEV_POP
COLLATE	MODULE	STDDEV_SAMP
COLLECT	MULTISET	SUBMULTISET
CONVERT	NATIONAL	SUM
CORR	NATURAL	SYMMETRIC
CORRESPONDING	NCHAR	TABLESAMPLE
COVAR_POP	NCLOB	TIMEZONE_HOUR
COVAR_SAMP	NORMALIZE	TIMEZONE_MINUTE
CUBE	NULLIF	TRAILING
CUME_DIST	NUMERIC	TRANSLATE
CURRENT_DEFAULT_TRANSFORM_GROUP	OCTET_LENGTH	TRANSLATION
CURRENT_ROLE	ONLY	TREAT
CURRENT_TRANSFORM_GROUP_FOR_TYPE	OVERLAPS	TRUE
DEC	OVERLAY	UESCAPE
DECIMAL	PERCENT_RANK	UNKNOWN
DEREF	PERCENTILE_CONT	UNNEST
ELEMENT	PERCENTILE_DISC	UPPER
EXEC	POWER	VAR_POP
EXP	REAL	VAR_SAMP
FALSE	RECURSIVE	VARCHAR
FILTER	REF	VARYING
FLOAT	REGR_AVGX	WIDTH_BUCKET
FLOOR	REGR_AVGY	WINDOW
FUSION	REGR_COUNT	WITHIN

Chapter 7. Naming conventions

The following conventions apply when naming database manager objects, such as databases and tables:

- Character strings that represent names of database manager objects can contain any of the following: a-z, A-Z, 0-9, @, #, and $.
- Unless otherwise noted, names can be entered in lowercase letters; however, the database manager processes them as if they were uppercase.

  The exception to this convention is character strings that represent names under the systems network architecture (SNA) which, as a communications protocol, is no longer supported. Many values are case sensitive, such as logical unit names (partner_lu and local_lu). The name must be entered exactly as it appears in the SNA definitions that correspond to those terms.
- A database name or database alias is a unique character string containing from one to eight letters, numbers, or keyboard characters from the set described above.

  Databases are cataloged in the system and local database directories by their aliases in one field, and their original name in another. For most functions, the database manager uses the name entered in the alias field of the database directories. (The exceptions are CHANGE DATABASE COMMENT and CREATE DATABASE, where a directory path must be specified.)
- The name or the alias name of a table or a view is an SQL identifier that is a unique character string 1 to 128 bytes in length. Column names can be 1 to 128 bytes in length.

  A fully qualified table name consists of the *schema.tablename*. The schema is the unique user ID under which the table was created. The schema name for a declared temporary table must be SESSION.
- Local aliases for remote nodes that are to be cataloged in the node directory cannot exceed eight characters in length.
- The first character in the string must be an alphabetic character, @, #, or $; it cannot be a number or the letter sequences SYS, DBM, or IBM.

The following conventions apply when naming user IDs and authentication IDs:

- Character strings that represent names of database manager objects can contain any of the following: a-z, A-Z, 0-9, @, #, and $.
- User IDs and groups may also contain any of the following additional characters when supported by the security plug-in: _, !, %, (,), {, }, –, ., ^.
- User IDs and groups containing any of the following characters must be delimited with quotations when entered through the command line processor: !, %, (,), {, }, –, ., ^,
- The first character in the string must be an alphabetic character, @, #, or $; it cannot be a number or the letter sequences SYS, DBM, or IBM.
- Authentication IDs cannot exceed 128 bytes in length.
- Group IDs cannot exceed 128 bytes in length.

Part 2. Partitioned database environments

Chapter 8. Parallel database systems

Partitioned database environments

The Database Partitioning Feature (DPF) extends the database manager to the parallel, multi-partition environment.

- 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.
- A *single-partition database* is a database having only one database partition. All data in the database is stored in that single database partition. In this case database partition groups, while present, provide no additional capability.
- A *multi-partition database* is a database with two or more database partitions. Tables can be located in one or more database partitions. When a table is in a database partition group consisting of multiple database partitions, some of its rows are stored in one database partition, and other rows are stored in other database partitions.

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

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

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

The database manager allows you to store data across several database partitions in the database. This means that the data is physically stored across more than one database partition, and yet can be accessed as though it were located in the same place. Applications and users accessing data in a multi-partition database do not need to be aware of the physical location of the data.

The data, while physically split, is used and managed as a logical whole. Users can choose how to distribute their data by declaring distribution keys. Users can also determine across which and over how many database partitions their data is distributed by selecting the table space and the associated database partition group in which the data should be stored. Suggestions for distribution and replication can be done using the DB2 Design Advisor. In addition, an updatable distribution map is used with a hashing algorithm to specify the mapping of distribution key values to database partitions, which determines the placement and retrieval of each row of data. As a result, you can spread the workload across a multi-partition database for large tables, while allowing smaller tables to be stored on one or more database

partitions. Each database partition has local indexes on the data it stores, resulting in increased performance for local data access.

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

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

A non-root installation of a DB2 database product does not support database partitioning. As a result, an add node operation cannot be run. You should not manually update the `db2nodes.cfg` file. A manual update will result in the display of a SQL6031N error.

Parallelism

Components of a task, such as a database query, can be run in parallel to dramatically enhance performance. The nature of the task, the database configuration, and the hardware environment, all determine how the DB2 database product will perform a task in parallel.

These considerations are interrelated, and should be considered together when you work on the physical and logical design of a database. The following types of parallelism are supported by the DB2 database system:

- I/O
- Query
- Utility

Input/output parallelism

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

Query parallelism

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

Interquery parallelism refers to the ability of the database to accept queries from multiple applications at the same time. Each query runs independently of the others, but the database manager runs all of them at the same time. DB2 database products have always supported this type of parallelism.

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

Intrapartition parallelism

Intrapartition parallelism refers to the ability to break up a query into multiple parts. Some DB2 utilities also perform this type of parallelism.

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

Figure 6 shows a query that is broken into four pieces that can be run in parallel, with the results returned more quickly than if the query were run in serial fashion. The pieces are copies of each other. To utilize intrapartition parallelism, you must configure the database appropriately. You can choose the degree of parallelism or let the system do it for you. The degree of parallelism represents the number of pieces of a query running in parallel.

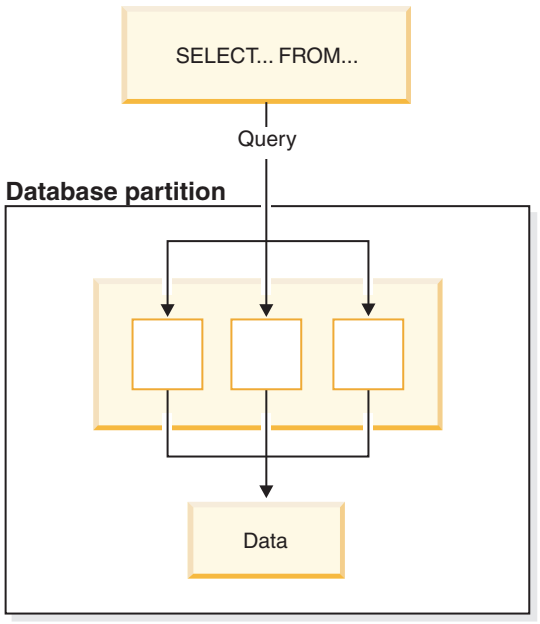


Figure 6. Intrapartition parallelism

Interpartition parallelism

Interpartition parallelism refers to the ability to break up a query into multiple parts across multiple partitions of a partitioned database, on one machine or multiple machines. The query is run in parallel. Some DB2 utilities also perform this type of parallelism.

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

Figure 7 on page 36 shows a query that is broken into four pieces that can be run in parallel, with the results returned more quickly than if the query were run in serial fashion on a single database partition.

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

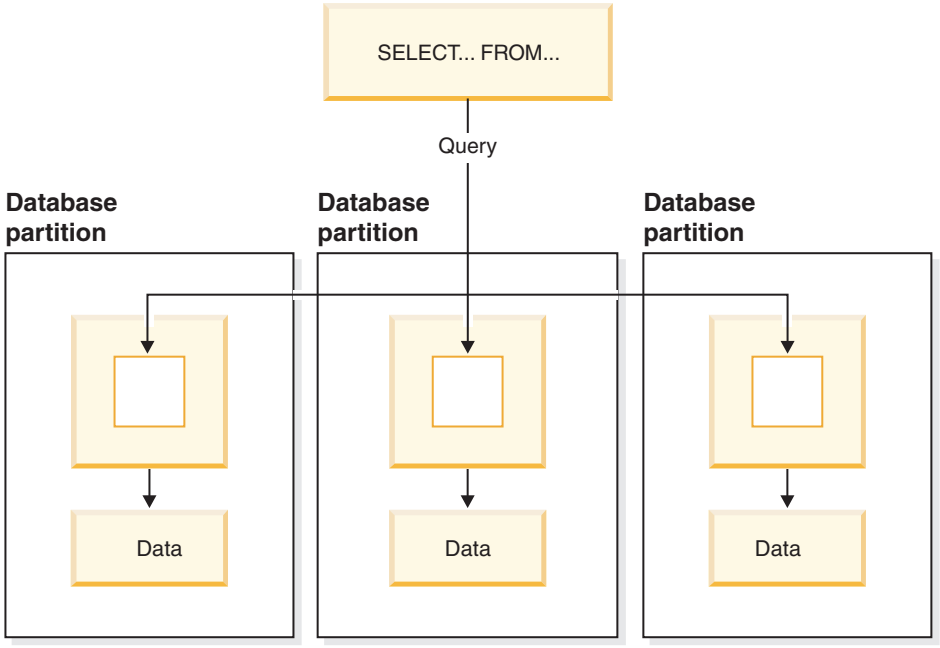


Figure 7. Interpartition parallelism

Simultaneous intrapartition and interpartition parallelism

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

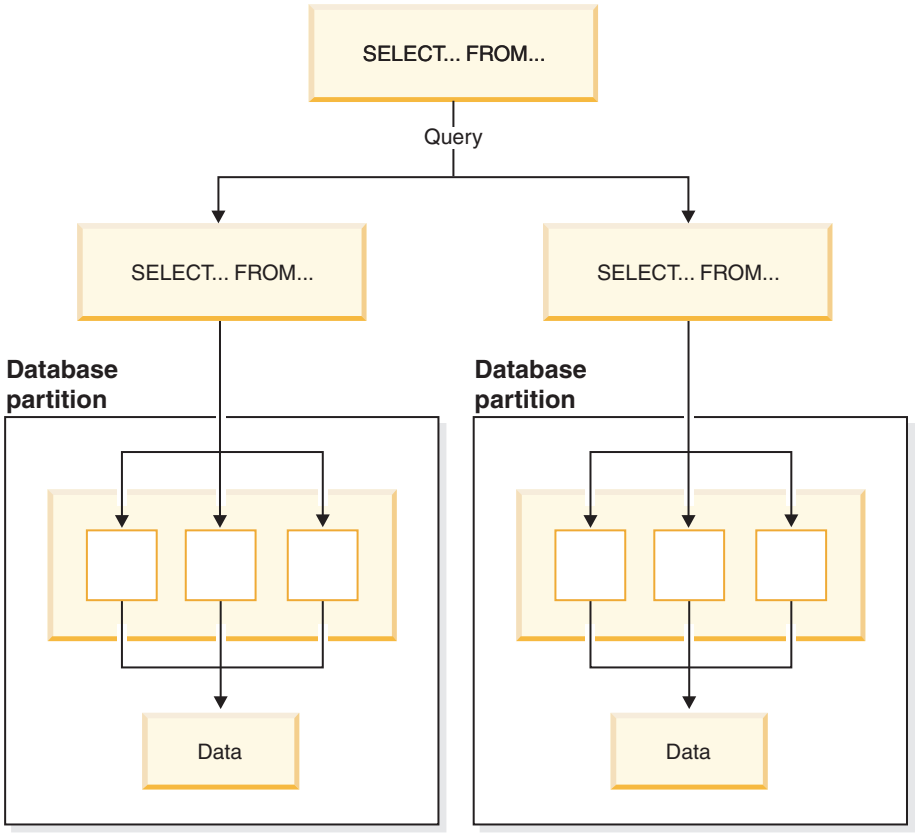


Figure 8. Simultaneous interpartition and intrapartition parallelism

Utility parallelism

DB2 utilities can take advantage of intrapartition parallelism. They can also take advantage of interpartition parallelism; where multiple database partitions exist, the utilities execute in each of the database partitions in parallel.

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

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

During index creation, the scanning and subsequent sorting of the data occurs in parallel. The DB2 system exploits both I/O parallelism and intrapartition parallelism when creating an index. This helps to speed up index creation when a CREATE INDEX statement is issued, during restart (if an index is marked invalid), and during the reorganization of data.

Backing up and restoring data are heavily I/O-bound tasks. The DB2 system exploits both I/O parallelism and intrapartition parallelism when performing backup and restore operations. Backup exploits I/O parallelism by reading from multiple table space containers in parallel, and asynchronously writing to multiple backup media in parallel.

Database partition and processor environments

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

This section provides an overview of the following hardware environments:

- Single database partition on a single processor (uniprocessor)
- Single database partition with multiple processors (SMP)
- Multiple database partition configurations
  - Database partitions with one processor (MPP)
  - Database partitions with multiple processors (cluster of SMPs)
  - Logical database partitions

Capacity and scalability are discussed for each environment.

Single database partition on a single processor

This environment is made up of memory and disk, but contains only a single CPU (see Figure 9). It is referred to by many different names, including stand-alone database, client/server database, serial database, uniprocessor system, and single node or non-parallel environment.

The database in this environment serves the needs of a department or small office, where the data and system resources (including a single processor or CPU) are managed by a single database manager.

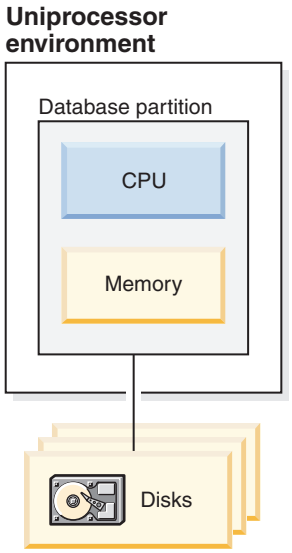


Figure 9. Single database partition on a single processor

Capacity and scalability

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

A single-processor system is restricted by the amount of disk space the processor can handle. As workload increases, a single CPU may not be able to process user

requests any faster, regardless of other components, such as memory or disk, that you may add. If you have reached maximum capacity or scalability, you can consider moving to a single database partition system with multiple processors.

Single database partition with multiple processors

This environment is typically made up of several equally powerful processors within the same machine (see Figure 10), and is called a *symmetric multiprocessor (SMP)* system. Resources, such as disk space and memory, are *shared*.

With multiple processors available, different database operations can be completed more quickly. DB2 database systems can also divide the work of a single query among available processors to improve processing speed. Other database operations, such as loading data, backing up and restoring table spaces, and creating indexes on existing data, can take advantage of multiple processors.

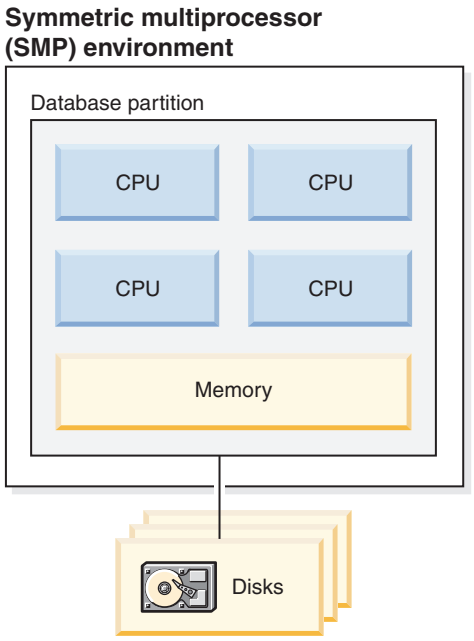


Figure 10. Single partition database symmetric multiprocessor environment

Capacity and scalability

In this environment you can add more processors. However, since the different processors may attempt to access the same data, limitations with this environment can appear as your business operations grow. With shared memory and shared disks, you are effectively sharing all of the database data.

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

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

Multiple database partition configurations

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

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

Database partitions with one processor

In this environment, there are many database partitions. Each database partition resides on its own machine, and has its own processor, memory, and disks (Figure 11). All the machines are connected by a communications facility. This environment is referred to by many different names, including: cluster, cluster of uniprocessors, massively parallel processing (MPP) environment, and shared-nothing configuration. The latter name accurately reflects the arrangement of resources in this environment. Unlike an SMP environment, an MPP environment has no shared memory or disks. The MPP environment removes the limitations introduced through the sharing of memory and disks.

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

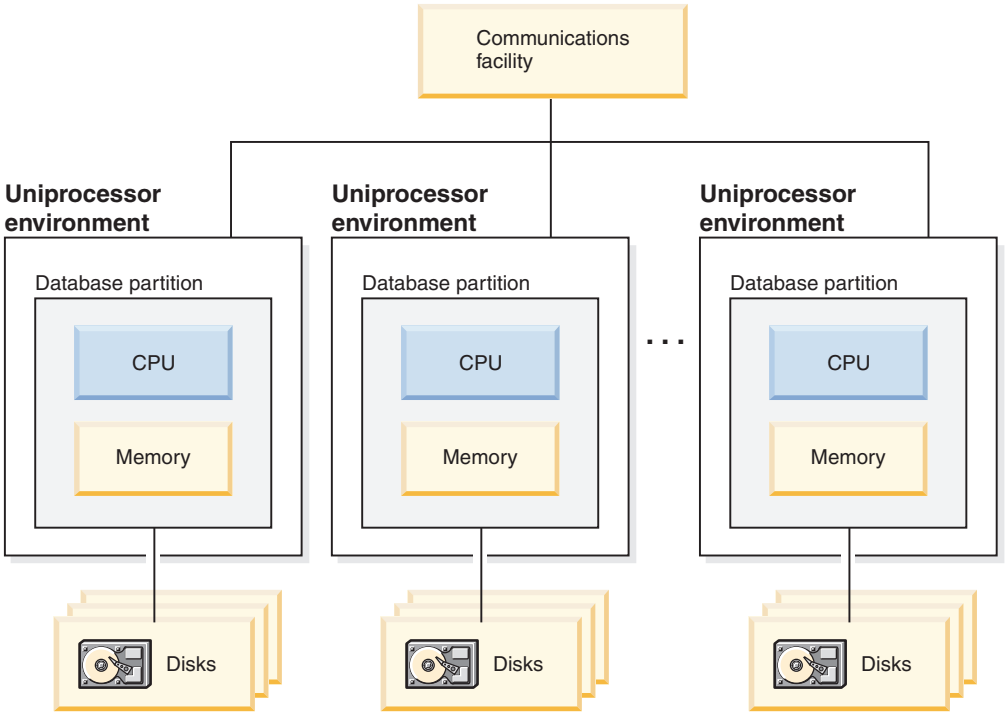


Figure 11. Massively parallel processing (MPP) environment

Capacity and scalability

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

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

Database partitions with multiple processors

An alternative to a configuration in which each database partition has a single processor, is a configuration in which each database partition has multiple processors. This is known as an *SMP cluster* (Figure 12).

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

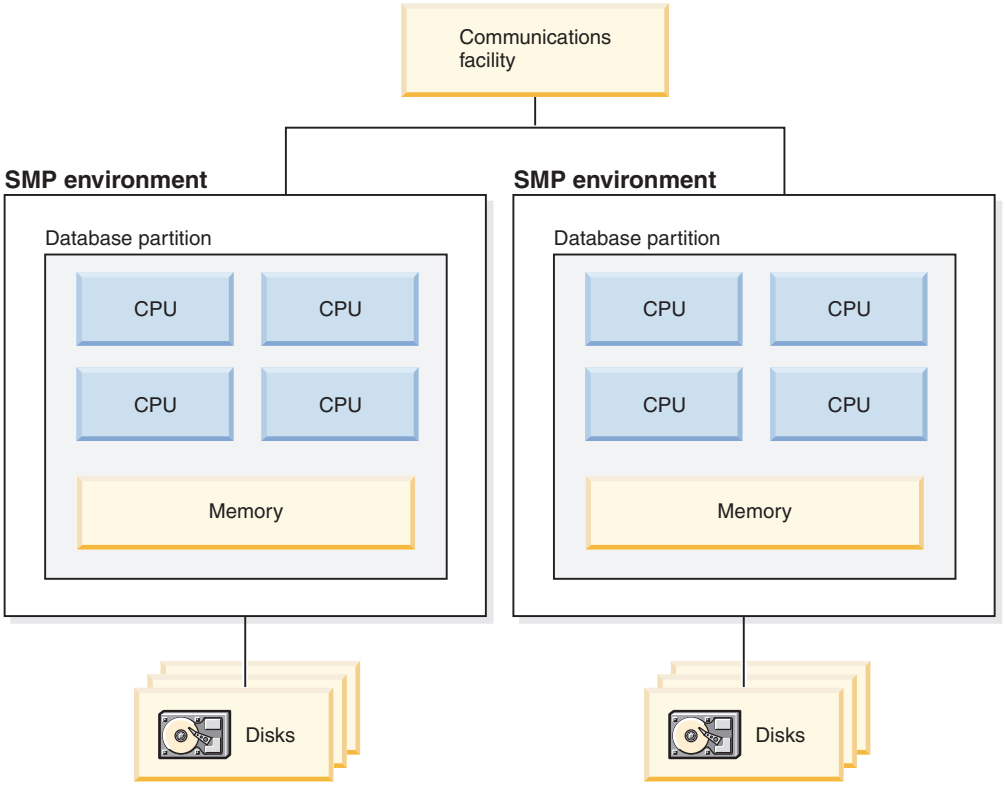


Figure 12. Several symmetric multiprocessor (SMP) environments in a cluster

Capacity and scalability

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

Logical database partitions

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

Logical database partitions provide scalability. Multiple database managers running on multiple logical partitions may make fuller use of available resources than a single database manager could. Figure 13 illustrates the fact that you may gain more scalability on an SMP machine by adding more database partitions; this is particularly true for machines with many processors. By distributing the database, you can administer and recover each database partition separately.

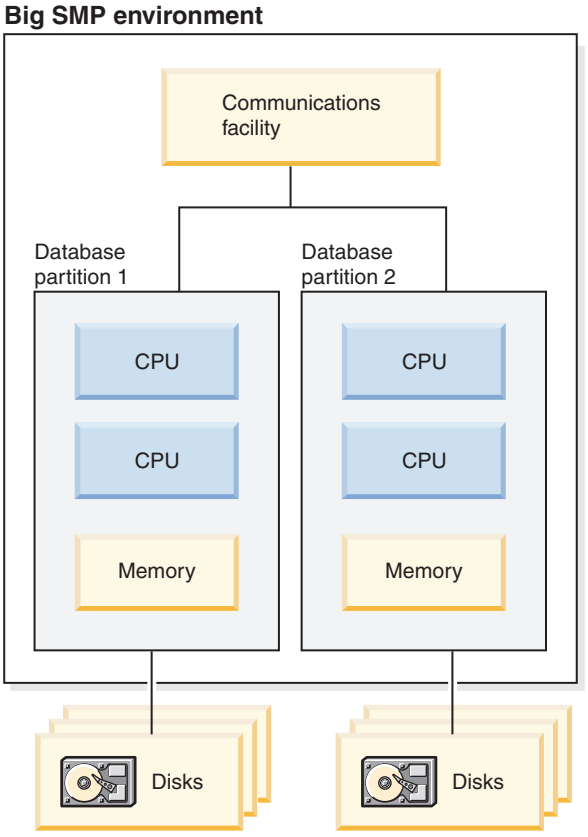


Figure 13. Partitioned database with symmetric multiprocessor environment

Figure 14 on page 43 illustrates that you can multiply the configuration shown in Figure 13 to increase processing power.

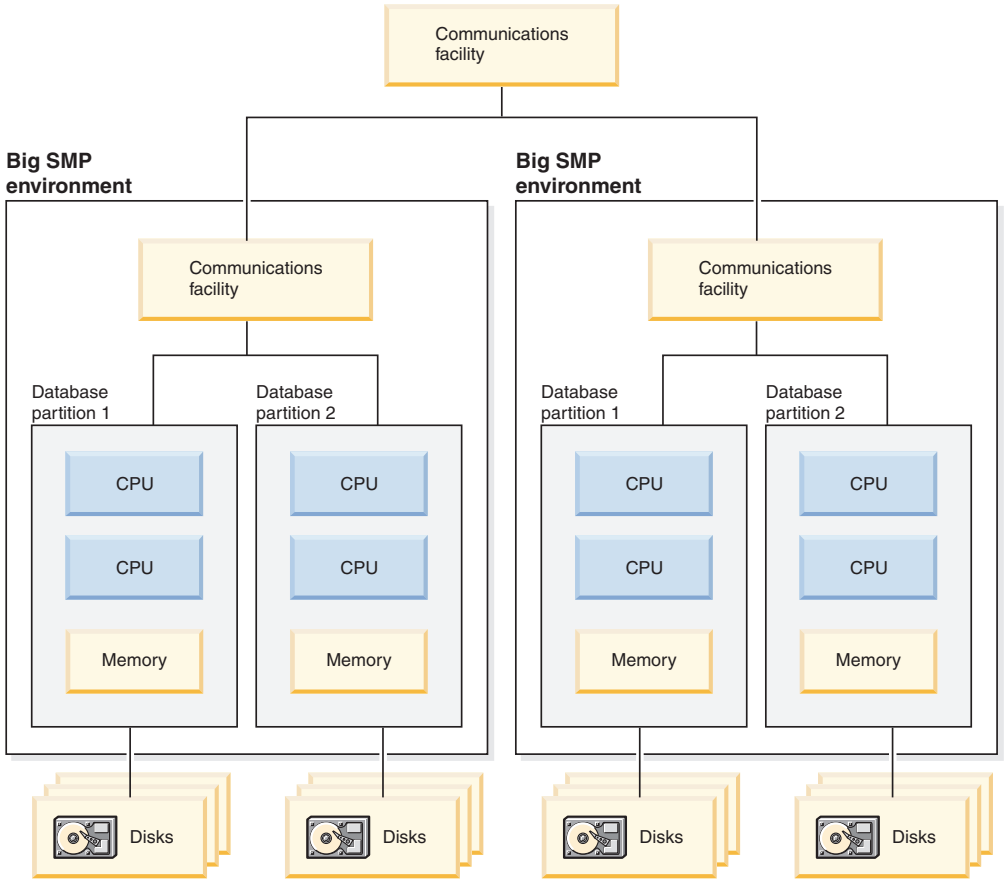


Figure 14. Partitioned database with symmetric multiprocessor environments clustered together

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

Summary of parallelism best suited to each hardware environment

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

Table 6. Types of Parallelism Possible in Each Hardware Environment

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

Table 6. Types of Parallelism Possible in Each Hardware Environment (continued)

Hardware Environment	I/O Parallelism	Intra-Query Parallelism	
		Intra-Partition Parallelism	**Inter-Partition Parallelism**
Multiple Database Partitions, Multiple Processors (cluster of SMPs)	Yes	Yes	Yes
Logical Database Partitions	Yes	Yes	Yes
Note: (1) There may be an advantage to setting the degree of parallelism (using one of the configuration parameters) to some value greater than one, even on a single processor system, especially if the queries you execute are not fully utilizing the CPU (for example, if they are I/O bound).			

Chapter 9. Database partitioning across multiple database partitions

The database manager allows great flexibility in spreading data across multiple database partitions (nodes) of a partitioned database. Users can choose how to distribute their data by declaring distribution keys, and can determine which and how many database partitions their table data can be spread across by selecting the database partition group and table space in which the data should be stored.

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

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

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

The distribution key for a table in a table space on a partitioned database partition group is specified in the CREATE TABLE statement or the ALTER TABLE statement. If not specified, a distribution key for a table is created by default from the first column of the primary key. If no primary key is defined, the default distribution key is the first column defined in that table that has a data type other than a long or a LOB data type. Tables in partitioned databases must have at least one column that is neither a long nor a LOB data type. A table in a table space that is in a single partition database partition group will have a distribution key only if it is explicitly specified.

Rows are placed in a database partition as follows:

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

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

The database manager has the capability of recognizing when data being accessed for a join or a subquery is located at the same database partition in the same database partition group. This is known as *table collocation*. Rows in collocated tables with the same distribution key values are located on the same database

partition. The database manager can choose to perform join or subquery processing at the database partition in which the data is stored. This can have significant performance advantages.

Collocated tables must:

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

Chapter 10. Database partition groups

A database partition group is a set of one or more database partitions defined as belonging to a database. When you want to create tables for the database, you first create the database partition group where the table spaces will be stored, then you create the table space where the tables will be stored.

You can define named subsets of one or more database partitions in a database. Each subset you define is known as a *database partition group*. Each subset that contains more than one database partition is known as a *multiple partition database partition group*. Multiple partition database partition groups can only be defined with database partitions that belong to the same instance. A database partition group can contain as few as one database partition, or span all of the database partitions in the database.

Figure 15 shows an example of a database with five database partitions in which:

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

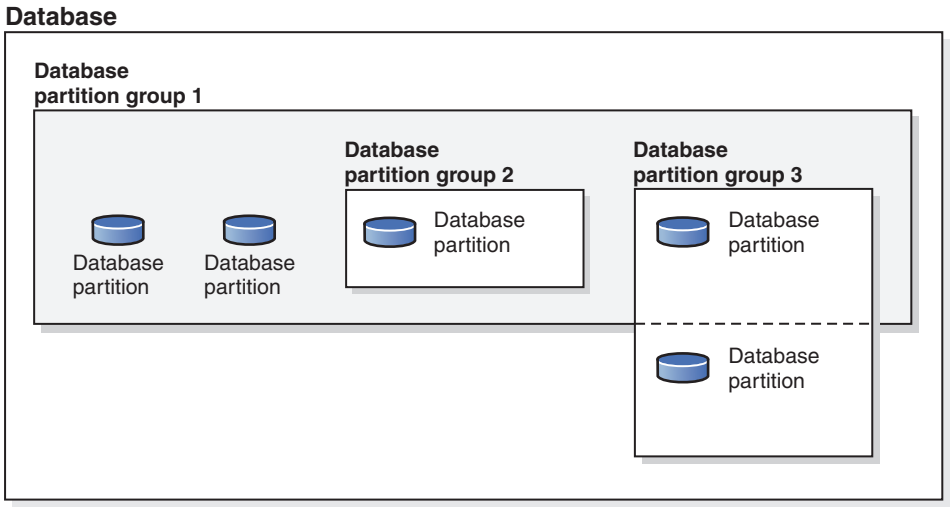


Figure 15. Database partition groups in a database

You create a new database partition group using the CREATE DATABASE PARTITION GROUP statement. You can modify it using the ALTER DATABASE PARTITION GROUP statement. Data is divided across all the database partitions in a database partition group, and you can add or drop one or more database

partitions from a database partition group. If you are using a multiple partition database partition group, you must look at several database partition group design considerations.

Each database partition that is part of the database system configuration must already be defined in a *database partition configuration file* called `db2nodes.cfg`. A database partition group can contain as few as one database partition, or as many as the entire set of database partitions defined for the database system.

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

In a non-partitioned database, no distribution key or distribution map is required. A database partition is a part of the database, complete with user data, indexes, configuration files, and transaction logs. Default database partition groups that were created when the database was created are used by the database manager. IBMCATGROUP is the default database partition group for the table space containing the system catalogs. IBMTEMPGROUP is the default database partition group for system temporary table spaces. IBMDEFAULTGROUP is the default database partition group for the table spaces containing the user defined tables that you may choose to put there. A user temporary table space for a declared temporary table or a created temporary table can be created in IBMDEFAULTGROUP or any user-created database partition group but not in IBMTEMPGROUP.

When working with database partition groups you can:

- Create a database partition group.
- Change the comment associated with a database partition group.
- Add database partitions to a database partition group.
- Drop database partitions from a database partition group.
- Redistribute table data within a database partition group.

Chapter 11. Execution parallelism

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

Enabling inter-partition query parallelism

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

About this task

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

Enabling parallelism for loading data

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

- CPU_PARALLELISM
- DISK_PARALLELISM

In a partitioned database environment, inter-partition parallelism for data loading occurs automatically when the target table is defined on multiple database partitions. Inter-partition parallelism for data loading can be overridden by specifying OUTPUT_DBPARTNUMS. The load utility also intelligently enables database partitioning parallelism depending on the size of the target database partitions. MAX_NUM_PART_AGENTS can be used to control the maximum degree of parallelism selected by the load utility. Database partitioning parallelism can be overridden by specifying PARTITIONING_DBPARTNUMS when ANYORDER is also specified.

Enabling parallelism when creating indexes

To enable parallelism when creating an index:

- The table must be large enough to benefit from parallelism
- Multiple processors must be enabled on an SMP computer.

Enabling I/O parallelism when backing up a database or table space

To enable I/O parallelism when backing up a database or table space:

1. Use more than one target media.
2. Configure table spaces for parallel I/O by defining multiple containers, or use a single container with multiple disks, and the appropriate use of the DB2_PARALLEL_IO registry variable. If you want to take advantage of parallel I/O, you must consider the implications of what must be done before you define any containers. This cannot be done whenever you see a need; it must be planned for before you reach the point where you need to backup your database or table space.
3. Use the PARALLELISM parameter on the BACKUP command to specify the degree of parallelism.

4. Use the WITH `num-buffers` BUFFERS parameter on the BACKUP command to ensure enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.

   Also, use a backup buffer size that is:

   - As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.
   - At least as large as the largest (extentsize * number of containers) product of the table spaces being backed up.

Enabling I/O parallelism when restoring a database or table space

To enable I/O parallelism when restoring a database or table space:

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

  Also, use a restore buffer size that is:

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

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 intra-partition parallelism enabled when **Connection concentrator** is enabled, this value limits the number of coordinating agents.

Chapter 12. Synchronizing clocks in a partitioned database environment

You should maintain relatively synchronized system clocks across the database partition servers to ensure smooth database operations and unlimited forward recoverability. Time differences among the database partition servers, plus any potential operational and communications delays for a transaction should be less than the value specified for the *max_time_diff* (maximum time difference among nodes) database manager configuration parameter.

To ensure that the log record time stamps reflect the sequence of transactions in a partitioned database environment, DB2 uses the system clock and the virtual timestamp stored in the `SQLOGCTL.LFH` file on each machine as the basis for the time stamps in the log records. If, however, the system clock is set ahead, the log clock is automatically set ahead with it. Although the system clock can be set back, the clock for the logs cannot, and remains at the *same* advanced time until the system clock matches this time. The clocks are then in synchrony. The implication of this is that a short term system clock error on a database node can have a long lasting effect on the time stamps of database logs.

For example, assume that the system clock on database partition server A is mistakenly set to November 7, 2005 when the year is 2003, and assume that the mistake is corrected *after* an update transaction is committed in the database partition at that database partition server. If the database is in continual use, and is regularly updated over time, any point between November 7, 2003 and November 7, 2005 is virtually unreachable through rollforward recovery. When the COMMIT on database partition server A completes, the time stamp in the database log is set to 2005, and the log clock remains at November 7, 2005 until the system clock matches this time. If you attempt to roll forward to a point in time within this time frame, the operation will stop at the first time stamp that is beyond the specified stop point, which is November 7, 2003.

Although DB2 cannot control updates to the system clock, the *max_time_diff* database manager configuration parameter reduces the chances of this type of problem occurring:

- The configurable values for this parameter range from 1 minute to 24 hours.
- When the first connection request is made to a non-catalog partition, the database partition server sends its time to the catalog partition for the database. The catalog partition then checks that the time on the database partition requesting the connection, and its own time are within the range specified by the *max_time_diff* parameter. If this range is exceeded, the connection is refused.
- An update transaction that involves more than two database partition servers in the database must verify that the clocks on the participating database partition servers are in synchrony before the update can be committed. If two or more database partition servers have a time difference that exceeds the limit allowed by *max_time_diff*, the transaction is rolled back to prevent the incorrect time from being propagated to other database partition servers.

Chapter 13. Restrictions on data redistribution

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

The following restrictions apply to data redistribution:

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

  - The partitioned table has an access mode of FULL ACCESS in the systables.access_mode catalog table.
  - The partitioned table does not have any partitions currently being attached or detached.
- For replicated materialized query tables, if the data in a database partition group contains replicated materialized query tables, you must drop these tables before you redistribute the data. After data is redistributed, you can recreate the materialized query tables.
- For database partitions that contain multi-dimensional-clustered tables (MDCs) use of the REDISTRIBUTE DATABASE PARTITION GROUP command is restricted and will not proceed successfully if there are any multi-dimensional-clustered tables in the database partition group that contain rolled out blocks that are pending cleanup. These MDC tables must be cleaned up before data redistribution can be resumed or restarted.
- Dropping tables that are currently marked in the DB2 catalog views as being in the state "Redistribute in Progress" is restricted. To drop a table in this state, first run the REDISTRIBUTE DATABASE PARTITION GROUP utility with the ABORT or CONTINUE option and an appropriate table list so that redistribution of the table is either completed or aborted.

Chapter 14. Scenario: Partitioning data in a database

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

Scenario:

A database DBPG1 has 2 database partitions, specified as (0, 1) and a database partition group definition (0, 1).

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

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

Distribution of data between the database partitions in DBPG1:

To add three new database partitions to the database, issue the following commands:

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

The following redistribute command will change the DBPG_1 definition from (0, 1) to (0, 1, 3, 4, 5) and redistribute the data as well:

```
DB2 REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 UNIFORM ADD DBPARTITIONNUM (3 TO 5) STOP AT 2006-03-10-07.00.00.000000;
```

Let us presume that the command ran successfully for tables T1, T2 and T3, and then stopped due to the specification of the STOP AT option.

To abort the data redistribution for the database partition group and to revert the changes made to tables T1, T2, and T3, issue the following command:

```
DB2 REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD
 RECOVERABLE ABORT;
```

You might abort the data redistribution if an error or an interruption occurred during the data redistribution and you do not wish to continue the redistribute operation. For this scenario, presume that this command was run successfully and that tables T1 and T2 were reverted to their original state.

To redistribute T5 and T4 only with 5000 4K pages as DATA BUFFER:

```
DB2 REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 UNIFORM ADD DBPARTITIONNUM (3 TO 5) TABLE (T5, T4) ONLY  DATA BUFFER 5000;
```

If the command ran successfully, the data in tables T4 and T5 will have been redistributed successfully.

To complete the redistribution of data on table T1, T2, and T3 in a specified order, issue:

```
DB2 REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 CONTINUE TABLE (T1) FIRST;
```

Specifying TABLE (T1) FIRST forces the database manager to process table T1 first so that it can return to being online (read-only) before other tables. All other tables are processed in an order determined by the database manager.

Note:

- The ADD DBPARTITIONNUM option and the DROP DBPARTITIONNUM option do not need to be specified. Instead, the ALTER DATABASE PARTITION GROUP statement may be used to add or drop database partitions prior to the REDISTRIBUTE DATABASE PARTITION GROUP command being executed, in which case the REDISTRIBUTE DATABASE PARTITION GROUP command will not add or drop partitions, but will simply redistribute the data according to the specified options.
- It is strongly recommended that the user take an offline database backup of the database prior to executing the REDISTRIBUTE DATABASE PARTITION GROUP command. This action is not shown in the examples above.
- The REDISTRIBUTE DATABASE PARTITION GROUP command is not rollforward recoverable. For a full discussion of this issue, refer to the "REDISTRIBUTE DATABASE PARTITION GROUP command".
- After the REDISTRIBUTE DATABASE PARTITION GROUP command finishes, all the table spaces it accessed will be left in the BACKUP PENDING state. Such table spaces must be backed up before the tables they contain will be accessible for write activity.

The steps above illustrate how you can use variations of the redistribute command to redistribute data between database partitions.

Part 3. DB2 security considerations

Chapter 15. What's New

Authorities overview

Various administrative authorities exist at the instance level and at the database level. These administrative authorities group together certain privileges and authorities so that you can grant them to the users who are responsible for these tasks in your database installation.

Instance level authorities

Instance level authorities enable you to perform instance-wide functions, such as creating and upgrading databases, managing table spaces, and monitoring activity and performance on your instance. No instance-level authority provides access to data in database tables. The following diagram summarizes the abilities given by each of the instance level administrative authorities:

- SYSADM –for users managing the instance as a whole
- SYSCTRL –for users administering a database manager instance
- SYSMAINT –for users maintaining databases within an instance
- SYSMON –for users monitoring the instance and its databases

A user with a higher-level authority also has the abilities given by the lower level authorities. For example, a user with SYSCTRL authority can perform the functions of users with SYSMAINT and SYSMON authority as well.

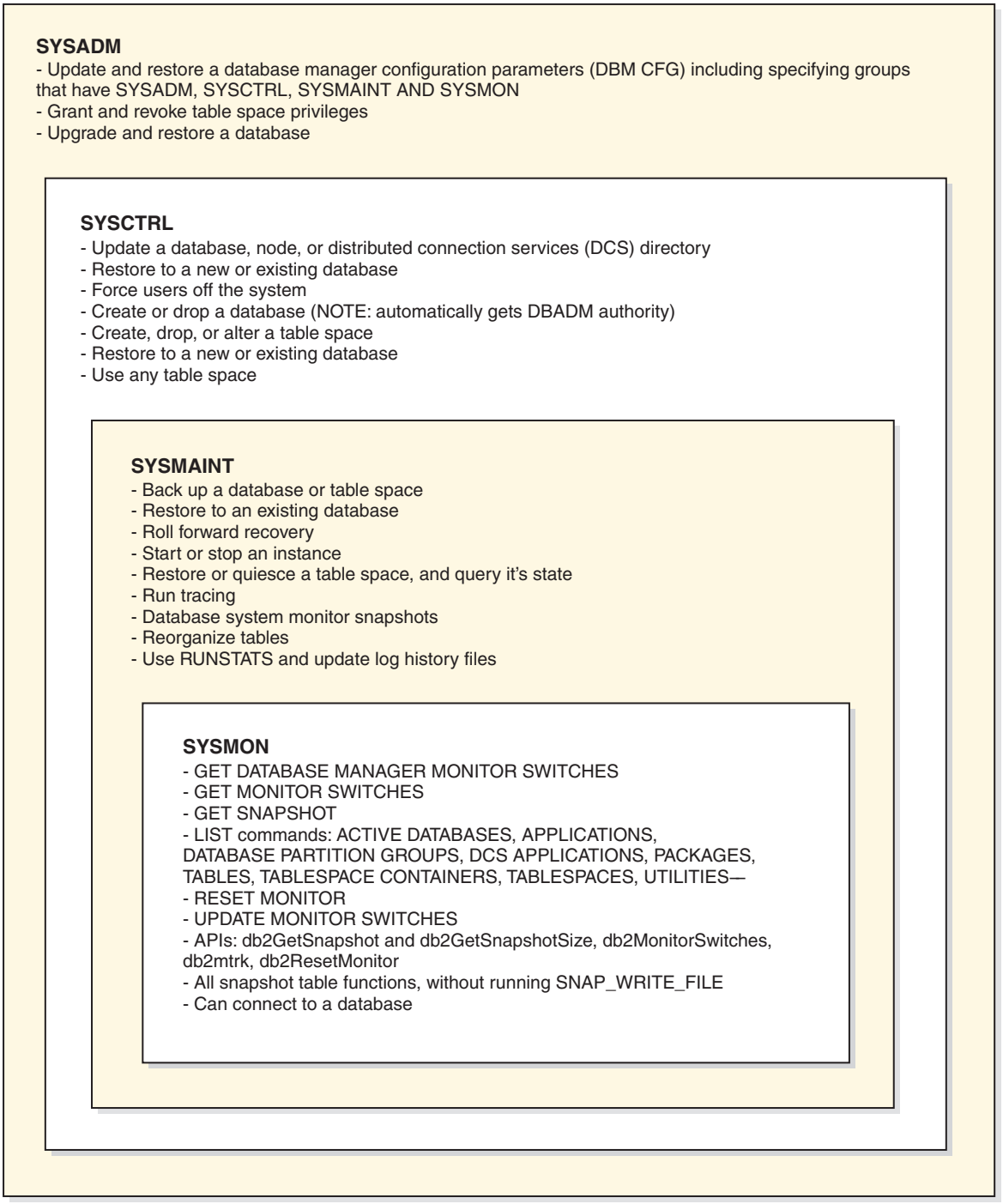


Figure 16. Instance-level authorities

Database level authorities

Database level authorities enable you to perform functions within a specific database, such as granting and revoking privileges, inserting, selecting, deleting and updating data, and managing workloads. The following diagram summarizes the abilities given by each of the database level authorities. The administrative database authorities are:

- SECADM – for users managing security within a database
- DBADM – for users administering a database

- ACCESSCTRL – for users who need to grant and revoke authorities and privileges (except for SECADM, DBADM, ACCESSCTRL, and DATAACCESS authority, SECADM authority is required to grant and revoke these authorities)
- DATAACCESS – for users who need to access data
- SQLADM – for users who monitor and tune SQL queries
- WLMADM – for users who manage workloads
- EXPLAIN – for users who need to explain query plans (EXPLAIN authority does not give access to the data itself)

The following diagram shows, where appropriate, which higher level authorities include the abilities given by a lower level authority. For example, a user with DBADM authority can perform the functions of users with SQLADM and EXPLAIN authority, and all functions except granting USAGE privilege on workloads, of users with WLMADM authority.

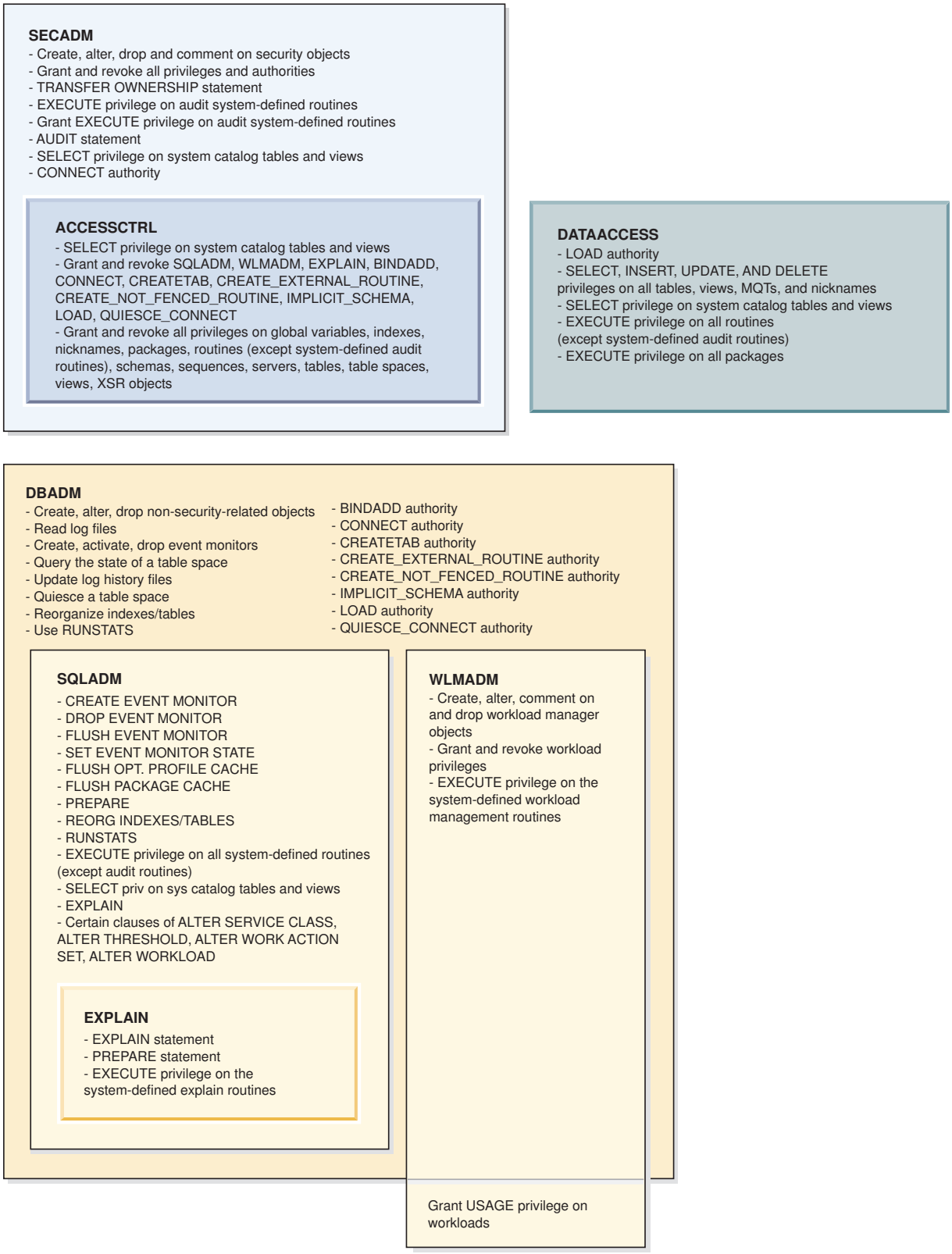


Figure 17. Database-level authorities

System administrator (SYSADM) authority scope has changed

In DB2 Version 9.7, the authorization model has been updated to clearly separate the duties of the system administrator, the database administrator, and the security administrator. As part of this enhancement, the abilities given by the SYSADM authority have been reduced.

Details

The changes for the SYSADM authority are as follows:

- A user who holds SYSADM authority no longer has implicit DBADM authority and therefore has limited capabilities compared to those available in Version 9.5. However, the UPGRADE DATABASE command and the RESTORE DATABASE command (for a downlevel database) grants DBADM authority to the SYSADM group. Privileges associated with groups are not considered for authorization when a user creates views, triggers, materialized query tables (MQTs), packages and SQL routines. Given these restrictions associated with groups, even though the upgrade process grants DBADM authority to the SYSADM group, the upgrade process alone does not ensure that every user with SYSADM authority in Version 9.5 will have the exact same capabilities in Version 9.7. For a member of the SYSADM group to be certain to retain the same privileges as in Version 9.5, they must be directly granted DBADM authority with DATAACCESS and ACCESSCTRL authorities, or must possess these authorities through membership of a role.
- If a user holding SYSADM authority creates a database, the user is automatically granted DATAACCESS, ACCESSCTRL, SECADM and DBADM authority for that database, which gives the user the same abilities as in Version 9.5.
- A user who holds SYSADM authority is no longer able to grant any authorities or privileges, except table space privileges.

User response

For a user holding SYSADM authority to obtain the same capabilities as in Version 9.5 (other than the ability to grant SECADM authority), the security administrator must explicitly grant the user DBADM authority and grant the user the new DATAACCESS and ACCESSCTRL authorities. The new authorities can be granted by using the GRANT DBADM ON DATABASE statement with the WITH DATAACCESS and WITH ACCESSCTRL options of that statement, which are default options. The DATAACCESS authority is the authority that allows access to data within a specific database, and the ACCESSCTRL authority is the authority that allows a user to grant and revoke privileges within a specific database.

For the user holding SYSADM authority to also be able to grant SECADM authority, the security administrator must grant the user SECADM authority as well. However, holding SECADM authority allows the user to perform more actions than the user could as a Version 9.5 system administrator. For example, the user can create objects such as roles, trusted contexts, and audit policies.

Tip: In addition to considering how these SYSADM authority changes impact your security implementation, you should also review the new capabilities of the database administrator (who holds DBADM authority) and the security administrator (who holds SECADM authority) , and the new authorities introduced in DB2 Version 9.7, so that you can decide how to organize responsibilities within your system. DB2 Version 9.7 introduces the following new authorities in addition to DATAACCESS and ACCESSCTRL:

- WLMADM, for managing workloads
- SQLADM, for tuning SQL statements
- EXPLAIN, for using the explain facility with SQL statements

These new authorities allow you to grant users responsibilities without granting them DBADM authority or privileges on base tables, which would give those users more privileges than they need to do their work.

Considerations for the Windows LocalSystem account

On Windows systems, when the **sysadm_group** database manager configuration parameter is not specified, the LocalSystem account is considered a system administrator (holding SYSADM authority). Any DB2 application that is run by LocalSystem is affected by the change in scope of SYSADM authority in Version 9.7. These applications are typically written in the form of Windows services and run under the LocalSystem account as the service logon account. If there is a need for these applications to perform database actions that are no longer within the scope of SYSADM, you must grant the LocalSystem account the required database privileges or authorities. For example, if an application requires database administrator capabilities, grant the LocalSystem account DBADM authority using the GRANT (Database Authorities) statement. Note that the authorization ID for the LocalSystem account is SYSTEM.

Security administrator (SECADM) abilities have been extended

In DB2 Version 9.7, the authorization model has been updated to clearly separate the duties of the system administrator, the database administrator, and the security administrator. As part of this enhancement, the abilities given by the SECADM authority have been extended.

Details

The changes for the SECADM authority are as follows:

- A user who holds SECADM authority can now grant and revoke all authorities and privileges, including DBADM and SECADM.
- The security administrator can now grant SECADM authority to roles and groups. In Version 9.5, SECADM could be granted only to a user.
- The security administrator can delegate responsibility to run the audit stored procedures and table functions (AUDIT_ARCHIVE, AUDIT_LIST_LOGS, and AUDIT_DELIM_EXTRACT) by granting another user EXECUTE privilege on them.

User response

The security administrator can allow another user to grant and revoke authorities and privileges by granting that other user the new ACCESSCTRL authority. However, only the security administrator can grant SECADM, DBADM, and ACCESSCTRL authority. Also, only the security administrator can grant the new authority DATAACCESS, which enables a user to access data within a specific database.

In addition to considering how these SECADM authority changes impact your security implementation, you should also review the new capabilities of the system administrator (who holds SYSADM authority) and the database administrator (who holds DBADM authority) , and the new authorities introduced in DB2

Version 9.7, so that you can decide how you to organize responsibilities within your system. DB2 Version 9.7 introduces the following new authorities in addition to DATAACCESS and ACCESSCTRL:

- WLMADM, for managing workloads
- SQLADM, for tuning SQL statements
- EXPLAIN, for using the explain facility with SQL statements

These new authorities allow you to grant users responsibilities without granting them DBADM authority or privileges on base tables, which would give those users more privileges than they need to do their work.

Access control administration authority (ACCESSCTRL)

ACCESSCTRL authority is the authority required to grant and revoke privileges on objects within a specific database. ACCESSCTRL authority has no inherent privilege to access data stored in tables, except the catalog tables and views.

ACCESSCTRL authority can only be granted by the security administrator (who holds SECADM authority). It can be granted to a user, a group, or a role. PUBLIC cannot obtain the ACCESSCTRL authority either directly or indirectly. ACCESSCTRL authority gives a user the ability to perform the following operations:

- Grant and revoke the following administrative authorities:
  - EXPLAIN
  - SQLADM
  - WLMADM
- Grant and revoke the following database authorities:
  - BINDADD
  - CONNECT
  - CREATETAB
  - CREATE_EXTERNAL_ROUTINE
  - CREATE_NOT_FENCED_ROUTINE
  - IMPLICIT_SCHEMA
  - LOAD
  - QUIESCE_CONNECT
- Grant and revoke all privileges on the following objects, regardless who granted the privilege:
  - Global Variable
  - Index
  - Nickname
  - Package
  - Routine (except audit routines)
  - Schema
  - Sequence
  - Server
  - Table
  - Table Space
  - View
  - XSR Objects
- SELECT privilege on the system catalog tables and views

This authority is a subset of security administrator (SECADM) authority.

Data access administration authority (DATAACCESS)

DATAACCESS is the authority that allows access to data within a specific database.

DATAACCESS authority can be granted only by the security administrator (who holds SECADM authority). It can be granted to a user, a group, or a role. PUBLIC cannot obtain the DATAACCESS authority either directly or indirectly.

For all tables, views, materialized query tables, and nicknames it gives these authorities and privileges:

- LOAD authority on the database
- SELECT privilege (including system catalog tables and views)
- INSERT privilege
- UPDATE privilege
- DELETE privilege

In addition, DATAACCESS authority provides the following privileges:

- EXECUTE on all packages
- EXECUTE on all routines (except audit routines)

Database administrator (DBADM) authority scope has changed

In DB2 Version 9.7, the authorization model has been updated to clearly separate the duties of the system administrator, the database administrator, and the security administrator. As part of this enhancement, the abilities given to the DBADM authority have changed.

Details

The changes for the DBADM authority are as follows:

- DBADM authority no longer necessarily includes the ability to access data and to grant and revoke privileges for a database.
- Granting DBADM authority no longer additionally grants the following separate database authorities because they are already implicitly vested in the DBADM authority level.
  - BINDADD
  - CONNECT
  - CREATETAB
  - CREATE_EXTERNAL_ROUTINE
  - CREATE_NOT_FENCED_ROUTINE
  - IMPLICIT_SCHEMA
  - QUIESCE_CONNECT
  - LOAD

User response

The new DATAACCESS authority provides the ability to access data in a database, and the new ACCESSCTRL authority provides the ability to grant and revoke privileges and authorities. These authorities are granted by default when a security

administrator grants DBADM authority. Also, the security administrator can use the following options of the GRANT DBADM ON DATABASE statement to provide or not provide the ACCESSCTRL and DATAACCESS authorities:

- WITH ACCESSCTRL
- WITHOUT ACCESSCTRL
- WITH DATAACCESS
- WITHOUT DATAACCESS

Tip: In addition to considering how these DBADM authority changes impact your security implementation, you should also review the new capabilities of the system administrator (who holds SYSADM authority) and security administrator (who holds SECADM authority) , and the new authorities introduced in DB2 Version 9.7, so that you can decide how you to organize responsibilities within your system. DB2 Version 9.7 introduces the following new authorities in addition to DATAACCESS and ACCESSCTRL:

- WLMADM, for managing workloads
- SQLADM, for tuning SQL statements
- EXPLAIN, for using the explain facility with SQL statements

These new authorities allow you to grant users responsibilities without granting them DBADM authority or privileges on base tables, which would give those users more privileges than they need to do their work.

SQL administration authority (SQLADM)

SQLADM authority is the authority required to monitor and tune SQL statements.

SQLADM authority can be granted by the security administrator (who holds SECADM authority) or a user who possesses ACCESSCTRL authority. SQLADM authority can be granted to a user, a group, a role, or to PUBLIC. SQLADM authority gives a user the ability to perform the following functions:

- Execution of the following SQL statements:
  - CREATE EVENT MONITOR
  - DROP EVENT MONITOR
  - EXPLAIN
  - FLUSH EVENT MONITOR
  - FLUSH OPTIMIZATION PROFILE CACHE
  - FLUSH PACKAGE CACHE
  - PREPARE
  - REORG INDEXES/TABLE
  - RUNSTATS
  - SET EVENT MONITOR STATE
- Execution of certain clauses of the following workload manager SQL statements:
  - The following clauses of the ALTER SERVICE CLASS statement:
    - COLLECT AGGREGATE ACTIVITY DATA
    - COLLECT AGGREGATE REQUEST DATA
    - COLLECT REQUEST METRICS
  - The following clause of the ALTER THRESHOLD statement
    - WHEN EXCEEDED COLLECT ACTIVITY DATA

    .

  - The following clauses of the ALTER WORK ACTION SET statement that allow you to alter a work action:
    - ALTER WORK ACTION ... COLLECT ACTIVITY DATA
    - ALTER WORK ACTION ... COLLECT AGGREGATE ACTIVITY DATA
    - ALTER WORK ACTION ... WHEN EXCEEDED COLLECT ACTIVITY DATA
  - The following clauses of the ALTER WORKLOAD statement:
    - COLLECT ACTIVITY METRICS
    - COLLECT AGGREGATE ACTIVITY DATA
    - COLLECT LOCK TIMEOUT DATA
    - COLLECT LOCK WAIT DATA
    - COLLECT UNIT OF WORK DATA
- SELECT privilege on the system catalog tables and views
- EXECUTE privilege on all system-defined DB2 routines (except audit routines)

SQLADM authority is a subset of the database administrator (DBADM) authority.

EXPLAIN authority is a subset of the SQLADM authority.

Explain administration authority (EXPLAIN)

EXPLAIN authority is the authority required to explain query plans without gaining access to data for a specific database. This authority is a subset of the database administrator authority and has no inherent privilege to access data stored in tables.

EXPLAIN authority can be granted by the security administrator (who holds SECADM authority) or by a user who possesses ACCESSCTRL authority. The EXPLAIN authority can be granted to a user, a group, a role, or to PUBLIC. It gives the ability to execute the following SQL statements:

- EXPLAIN
- PREPARE
- DESCRIBE on output of a SELECT statement or of an XQuery statement

EXPLAIN authority also provides EXECUTE privilege on the system-defined explain routines.

EXPLAIN authority is a subset of the SQLADM authority.

Workload administration authority (WLMADM)

WLMADM authority is the authority required to manage workload objects for a specific database. This authority allows you to create, alter, drop, comment on, and grant and revoke access to workload manager objects.

WLMADM authority can be granted by the security administrator (who holds SECADM authority) or a user who possesses ACCESSCTRL authority. WLMADM authority can be granted to a user, a group, a role, or to PUBLIC. WLMADM authority gives a user the ability to perform the following operations:

- Create, alter, comment on, and drop the following workload manager objects:
  - Histogram templates
  - Service classes
  - Thresholds

  - Work action sets
  - Work class sets
  - Workloads
- Grant and revoke workload privileges
- Execute the system-defined workload management routines.

WLMADM authority is a subset of the database administrator authority, DBADM.

SSLconfig.ini and SSLClientconfig.ini files replaced with new database manager configuration parameters

You no longer need to use the `SSLconfig.ini` and `SSLClientconfig.ini` configuration files to set up SSL support. The parameters that you used to set in these files have been replaced with database manager configuration parameters.

Details

The new database manager configuration parameters for server-side SSL support are as follows:

- **ssl_svr_keydb** specifies the fully qualified path of the key database file.
- **ssl_svr_stash** specifies the fully qualified path of the stash file that holds the encrypted password to the key database.
- **ssl_svr_label** specifies the label of the digital certificate of the server in the key database.
- **ssl_svcename** specifies the port that the database server uses to await communications from remote clients using the SSL protocol.
- **ssl_cipherspecs** (optional) specifies the cipher suites that the server supports.
- **ssl_versions** (optional) specifies the SSL and TLS versions that the server supports.

The new database manager configuration parameters for client-side SSL support are as follows:

- **ssl_clnt_keydb** specifies the fully qualified path of the key database file on the client.
- **ssl_clnt_stash** specifies the fully qualified path of the stash file on the client.

User response

To set up SSL support, set values for the new database manager configuration parameters.

The following tables show how the parameters in the `SSLconfig.ini` and `SSLClientconfig.ini` files map to these new database manager configuration parameters. The **ssl_cipherspecs** and **ssl_versions** parameters do not have equivalent parameters in these files; they provide new configuration options.

Table 7. Mapping of server-side SSL support parameters to new database manager configuration parameters

Version 9.5 SSLconfig.ini parameters	Version 9.7 database manager configuration parameters
DB2_SSL_KEYSTORE_FILE	ssl_svr_keydb
DB2_SSL_KEYSTORE_PW	ssl_svr_stash

Table 7. Mapping of server-side SSL support parameters to new database manager configuration parameters (continued)

Version 9.5 SSLconfig.ini parameters	Version 9.7 database manager configuration parameters
DB2_SSL_KEYSTORE_LABEL	ssl_svr_label
DB2_SSL_LISTENER	ssl_svcename

The **ssl_svr_stash** database manager configuration parameter is not exactly equivalent to the **DB2_SSL_KEYSTORE_PW** parameter. The **ssl_svr_stash** configuration parameter points to a stash file that holds the encrypted password to a key database, whereas the **DB2_SSL_KEYSTORE_PW** parameter specifies the password itself.

Table 8. Mapping of client-side SSL support parameters to new database manager configuration parameters

Version 9.5 SSLClientconfig.ini parameters	Version 9.7 database manager configuration parameters
DB2_SSL_KEYSTORE_FILE	ssl_clnt_keydb
DB2_SSL_KEYRING_STASH_FILE	ssl_clnt_stash

Security enhancements

DB2 authorization model has been enhanced to allow separation of duties

Version 9.7 clearly divides the duties of the database administrator and the security administrator and introduces new authorities that enable you to grant only the access a user needs to do their work. These enhancements also make it easier to meet government compliance requirements.

Version 9.7 introduces new authorities for workload management (WLMADM), SQL tuning (SQLADM) and for using the explain facility with SQL statements (EXPLAIN). These authorities allow you to grant users these responsibilities without having to grant them DBADM authority or actual privileges on the base tables, which would give those users more privileges than they need to do their work. Therefore, by using these new authorities, you can minimize the risk of exposing sensitive data.

Version 9.7 also introduces the new authorities DATAACCESS and ACCESSCTRL. DATAACCESS authority is the authority that allows access to data within a specific database. ACCESSCTRL authority is the authority that allows a user to grant and revoke privileges on objects within a specific database. By default, DATAACCESS and ACCESSCTRL authorities are included when the security administrator grants DBADM authority. But if you do not want your database administrator to have access to data, or to be able to grant privileges and authorities, you can choose to not include these authorities.

Note: The creator of a database is automatically granted DBADM, SECADM, DATAACCESS and ACCESSCTRL authorities within that database. If you do not want this user to have any of these authorities, you must revoke them.

Changes for the system administrator (who holds SYSADM authority)

A user who holds SYSADM authority no longer has implicit DBADM authority, so has limited capabilities compared to those available in Version 9.5.

A user who holds SYSADM authority is no longer able to grant any authorities or privileges, except to grant table space privileges.

For a user holding SYSADM authority to obtain the same capabilities as in Version 9.5 (other than the ability to grant SECADM authority), the security administrator must explicitly grant them DBADM authority. Note that when the security administrator grants DBADM authority, the new DATAACCESS and ACCESSCTRL authorities are included by default. This will give the user equivalent Version 9.5 capability. For this user to also be able to grant SECADM authority, they must be granted SECADM authority as well. Note, however, that holding SECADM authority will allow this user to perform more actions than they could as a Version 9.5 system administrator. For example, they will be able to create objects such as roles, trusted contexts and audit policies.

On Windows systems, when the **sysadm_group** database manager configuration parameter is not specified, the LocalSystem account is considered a system administrator (holding SYSADM authority). Any DB2 application that is run by LocalSystem is affected by the change in scope of SYSADM authority in Version 9.7. These applications are typically written in the form of Windows services and run under the LocalSystem account as the service logon account. If there is a need for these applications to perform database actions that are no longer within the scope of SYSADM, you must grant the LocalSystem account the required database privileges or authorities. For example, if an application requires database administrator capabilities, grant the LocalSystem account DBADM authority using the GRANT (Database Authorities) statement. Note that the authorization ID for the LocalSystem account is SYSTEM.

Changes for the security administrator (who holds SECADM authority)

A user who holds SECADM authority can now grant and revoke all authorities and privileges including DBADM and SECADM authorities.

SECADM authority can now be granted to roles and groups (in Version 9.5, SECADM could be granted only to a user).

SECADM authority is no longer necessary to run the audit stored procedures and table functions:

- AUDIT_ARCHIVE
- AUDIT_LIST_LOGS
- AUDIT_DELIM_EXTRACT

In Version 9.7, EXECUTE privilege is sufficient to run these routines, however, only the security administrator can grant the EXECUTE privilege on these routines. This change allows the security administrator to delegate part of their responsibilities to other users.

Changes for the database administrator (who holds DBADM authority)

The following authorities will continue to be available to the database administrator as long as the user holds DBADM authority, but will be lost if DBADM authority is revoked. Granting DBADM authority no longer additionally grants the following separate database authorities because they are already implicitly vested in the DBADM authority level.

- BINDADD
- CONNECT
- CREATETAB
- CREATE_EXTERNAL_ROUTINE
- CREATE_NOT_FENCED_ROUTINE
- IMPLICIT_SCHEMA
- QUIESCE_CONNECT
- LOAD

When the security administrator grants DBADM authority, they can choose whether to give the database administrator the ability to perform the following operations:

- Accessing data within the database.
- Granting and revoking privileges and authorities.

The security administrator can use the following options of the GRANT DBADM ON DATABASE statement to control these functions:

- WITH ACCESSCTRL
- WITHOUT ACCCESSCTRL
- WITH DATAACCESS
- WITHOUT DATAACCCESS

By default, DATAACCESS and ACCESSCTRL authorities are included if they are not specified.

SYSMON authority has been extended to LIST commands and the db2mtrk command

To improve the database monitoring capability of a user holding system monitor (SYSMON) authority, SYSMON now includes the ability to run certain LIST commands. Also, SYSMON authority enables you to run the db2mtrk command to report memory pool allocation information.

The affected LIST commands are as follows:

- LIST DATABASE PARTITION GROUPS
- LIST DRDA INDOUBT TRANSACTIONS
- LIST PACKAGES
- LIST TABLES
- LIST TABLESPACE CONTAINERS
- LIST TABLESPACES
- LIST UTILITIES

SSL client support expanded and configuration simplified

In DB2 Version 9.7, enhanced support for Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS), improves the security of data communication by making it easier to configure your server. Additionally, support is expanded to all non-Java DB2 clients, such as CLI/ODBC, .Net Data Provider, embedded SQL, and CLP.

Note: In this topic, references to SSL also apply to TLS.

TLS version 1.0 (RFC2246) and TLS version 1.1 (RFC4346) are supported.

Configuration enhancements

You no longer need to use separate configuration files to set up SSL support. The parameters that you used to set in the `SSLconfig.ini` and `SSLClientconfig.ini` files are now replaced by database manager configuration parameters and connection string keywords.

- There are six new server-side configuration parameters:
  - **ssl_svr_keydb** specifies the fully qualified path of the key database file.
  - **ssl_svr_stash** specifies the fully qualified path of the stash file that holds the encrypted password to the key database.
  - **ssl_svr_label** specifies the label of the digital certificate of the server in the key database.
  - **ssl_svcename** specifies the port that the database server uses to await communications from remote clients using the SSL protocol.
  - **ssl_cipherspecs** (optional) specifies the cipher suites that the server supports.
  - **ssl_versions** (optional) specifies the SSL and TLS versions that the server supports.
- There are two new client-side database manager configuration parameters:
  - **ssl_clnt_keydb** specifies the fully qualified path of the key database file on the client.
  - **ssl_clnt_stash** specifies the fully qualified path of the stash file on the client.
- There are two new connection string keywords for CLI/ODBC applications:
  - **ssl_client_keystoredb** - Set **ssl_client_keystoredb** to the fully-qualified key database file name.
  - **ssl_client_keystash** - Set **ssl_client_keystash** to the fully-qualified stash file name.
- There are three new connection string keywords for DB2 .Net Data Provider applications:
  - **SSLClientKeystoredb** - Set **SSLClientKeystoredb** to the fully-qualified key database file name.
  - **SSLClientKeystash** - Set **SSLClientKeystash** to the fully-qualified stash file name.
  - **security** - Set **security** to `SSL`.

Setting up SSL connections for CLI/ODBC applications

If you are using the IBM Data Server Driver for ODBC and CLI to connect to a database using SSL, you use the connection string parameters

ssl_client_keystoredb, and **ssl_client_keystash** to set the path for the client key database and for the stash file, and the connection string parameter **security** to set the protocol to SSL.

If you are using the IBM Data Server Client or IBM Data Server Runtime Client to connect to a database using SSL, you use the connection string parameter **security** to set the protocol to SSL, and you can use either the connection string parameters **ssl_client_keystoredb** and **ssl_client_keystash**, or the client-side database manager configuration parameters **ssl_clnt_keydb** and **ssl_clnt_stash**, to set the path for the client key database and for the stash file.

Setting up SSL connections for .Net Data Provider applications

For .Net Data Provider applications, you use the connection string parameters **SSLClientKeystoredb** and **SSLClientKeystash** to set the path for the client key database and for the stash file, and the connection string parameter **security** to set the protocol to SSL

Setting up SSL connections for CLP clients and embedded SQL clients

The SSL keyword has been added to the CATALOG TCPIP NODE command SECURITY parameter. CLP clients and embedded SQL clients can use this keyword and the client-side database manager configuration parameters **ssl_clnt_keydb**, and **ssl_clnt_stash** to connect to a database using SSL.

security CLI/ODBC configuration keyword

Specifies whether or not SSL support is used for File DSN or in a DSN-less connection.

db2cli.ini keyword syntax:
security = SSL

Default setting:
None.

Usage notes:
This can be set in the `[Data Source]` section of the `db2cli.ini` file for the given data source, or in a connection string.

This parameter specifies whether a TCP/IP communication will be with SSL support or not. It can only be used with the protocols TCPIP, TCPIP4, or TCPIP6. If not specified, a normal TCP/IP without SSL support will be used.

SSL_client_keystoredb CLI/ODBC configuration keyword

Specifies the SSL key file that is used for File DSN or in a DSN-less connection.

db2cli.ini keyword syntax:
ssl_client_keystoredb = <*fully qualified key file path*>

Default setting:
None.

Usage notes:
This can be set in the `[Data Source]` section of the `db2cli.ini` file for the given data source, or in a connection string.

This parameter specifies the fully qualified path of the key file (.kdb). The key file stores the signer certificate from the server personal certificate.

- For a self-signed server personal certificate, the signer certificate is the public key of the personal certificate.
- For a certificate authority signed server personal certificate, the signer certificate is the root CA certificate of the CA that signed the personal certificate.

If the SSL protocol (**security**=SSL) is used, this parameter must be defined. The signer certificate from the server personal certificate must exist so that the client can authenticate the server, otherwise the connection fails.

SSL_client_keystash CLI/ODBC configuration keyword

Specifies the SSL stash file used for File DSN or in a DSN-less connection.

db2cli.ini keyword syntax:

ssl_client_keystash = *<fully qualified stash file path>*

Default setting:

None.

Usage notes:

This can be set in the `[Data Source]` section of the `db2cli.ini` file for the given data source, or in a connection string.

This parameter specifies the fully qualified path of the stash file (.sth), which stores an obfuscated version of the key database password. The file will be used to access the SSL key file during the SSL handshake. This parameter must be defined if the SSL protocol (**security**=SSL) is used, otherwise the connection fails.

AES encryption of user ID and password enhances security

In Version 9.7, you can now encrypt the user ID and password using the Advanced Encryption Standard (AES) algorithm with keys 256 bits long.

The user ID and password submitted for authentication to DB2 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 of the **authentication** configuration parameter on the server and the authentication requested by the client. The choice of the encryption algorithm used to encrypt the user ID and password, either DES or AES, depends on the setting of the **alternate_auth_enc** database manager configuration parameter:

- NOT_SPECIFIED (the default) means that the server accepts the encryption algorithm that the client proposes.
- AES_CMP means that if the connecting client proposes DES but supports AES encryption, the server renegotiates for AES encryption. Downlevel clients that do not support AES will still be able to connect using DES.
- AES_ONLY means that the server accepts only AES encryption. If the client does not support AES encryption, the connection is rejected.

Discontinued functionality

Type-1 indexes have been discontinued

Type-1 indexes are no longer supported. You must convert type-1 indexes to type-2 indexes.

Details

All indexes that you created by using DB2 releases earlier than Version 8 are type-1 indexes, unless you converted them to type-2 indexes in Version 8 or later via the REORG INDEXES command with the **CONVERT** option. All indexes that you created by using Version 8.2, Version 9.1, or Version 9.5 are type-2 indexes, unless they were created in an instance with the **DB2_INDEX_TYPE2** registry variable set to NO, or unless you created an index on a table that already had a type-1 index. In Version 9.7, all indexes that you create are type-2 indexes.

If you do not convert your type-1 indexes before upgrading a database, these indexes are marked as invalid during the upgrade. If you set the **indexrec** configuration parameter to RESTART, the indexes are rebuilt as type-2 indexes when you restart the database. Otherwise, the rebuild occurs when you first access a table, and you might experience an unexpected degradation in response time. The table is inaccessible until the index rebuild is completed.

Also, the following related functionality is deprecated and might be removed in a future release:

- The **CONVERT** option of the REORG INDEXES command
- The **DB2LOADQUERY_TYPE1_INDEXES** parameter of the db2LoadQueryOutputStruct data structure and of the db2LoadQueryOutputStruct64 data structure of the db2LoadQuery API
- The DB2REORG_CONVERT parameter of the db2ReorgStruct data structure of the db2Reorg API

User response

Before upgrading to DB2 Version 9.7, convert type-1 indexes to type-2 indexes. Ensure that you allocate enough time to convert all the indexes prior to upgrading.

You can convert type-1 indexes to type-2 indexes by using the **CONVERT** option of the REORG INDEXES command or by using the output of the db2IdentifyType1 command. The db2IdentifyType1 command identifies and generates the appropriate statements that you can use later to convert any type-1 indexes found in tables or schemas for a specified database. For more information, see the “Converting type-1 indexes to type-2 indexes” topic.

Chapter 16. Authentications, authorizations, privileges, and authorities

Security

To protect data and resources associated with a database server, the DB2 database manager uses a combination of external security services and internal access control information. To access a database server, you must pass some security checks before you are given access to database data or resources. The first step in database security is called *authentication*, where you must prove that you are who you say you are. The second step is called *authorization*, where the database manager decides if the validated user is allowed to perform the requested action, or access the requested data.

Authentication

Authentication of a user is completed using a security facility outside of the DB2 database system. The security facility can be part of the operating system or a separate product.

The security facility requires two items to authenticate a user: a user ID and a password. The user ID identifies the user to the security facility. By supplying the correct password, information known only to the user and the security facility, the user's identity (corresponding to the user ID) is verified.

Note: In non-root installations, operating system-based authentication must be enabled by running the db2rfe command.

After being authenticated:

- The user must be identified to DB2 using an SQL authorization name or *authid*. This name can be the same as the user ID, or a mapped value. For example, on UNIX operating systems, when you are using the default security plug-in module, a DB2 *authid* is derived by transforming to uppercase letters a UNIX user ID that follows DB2 naming conventions.
- A list of groups to which the user belongs is obtained. Group membership may be used when authorizing the user. Groups are security facility entities that must also map to DB2 authorization names. This mapping is done in a method similar to that used for user IDs.

The DB2 database manager uses the security facility to authenticate users in one of two ways:

- A successful security system login is used as evidence of identity, and allows:
  - Use of local commands to access local data
  - Use of remote connections when the server trusts the client authentication.
- Successful validation of a user ID and password by the security facility is used as evidence of identity and allows:
  - Use of remote connections where the server requires proof of authentication
  - Use of operations where the user wants to run a command under an identity other than the identity used for login.

Note: On some UNIX systems, the DB2 database manager can log failed password attempts with the operating system, and detect when a client has exceeded the number of allowable login tries, as specified by the LOGINRETRIES parameter.

Partitioned database authentication considerations

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

Consistency across all partitions is recommended.

Authorization

Authorization is performed using DB2 facilities. DB2 tables and configuration files are used to record the permissions associated with each authorization name.

When an authenticated user tries to access data, these recorded permissions are compared with the permissions of:

- The authorization name of the user
- The groups to which the user belongs
- The roles granted to the user directly or indirectly through a group or a role
- The permissions acquired through a trusted context

Based on this comparison, the DB2 server determines whether to allow the requested access.

The types of permissions recorded are privileges, authority levels, and LBAC credentials.

A *privilege* defines a single permission for an authorization name, enabling a user to create or access database resources. Privileges are stored in the database catalogs.

Authority levels provide a method of grouping privileges and control over database manager operations. Database-specific authorities are stored in the database catalogs; system authorities are associated with group membership, and the group names that are associated with the authority levels are stored in the database manager configuration file for a given instance.

LBAC credentials are LBAC security labels and LBAC rule exemptions that allow access to data protected by label-based access control (LBAC). LBAC credentials are stored in the database catalogs.

Groups provide a convenient means of performing authorization for a collection of users without having to grant or revoke privileges for each user individually. Unless otherwise specified, group authorization names can be used anywhere that authorization names are used for authorization purposes. In general, group membership is considered for dynamic SQL and non-database object authorizations (such as instance level commands and utilities), but is not considered for static SQL. The exception to this general case occurs when privileges are granted to PUBLIC: these are considered when static SQL is processed. Specific cases where group membership does not apply are noted throughout the DB2 documentation, where applicable.

A role is a database object that groups together one or more privileges and can be assigned to users, groups, PUBLIC, or other roles by using a GRANT statement or to a trusted context by using a CREATE TRUSTED CONTEXT or ALTER TRUSTED CONTEXT statement. A role can be specified for the SESSION_USER ROLE connection attribute in a workload definition. When you use roles, you associate access permissions on database objects with the roles. Users that are members of those roles then have the privileges defined for the role with which to access database objects.

Roles provide similar functionality as groups; they perform authorization for a collection of users without having to grant or revoke privileges for each user individually. One advantage of roles is that they are managed by the DB2 database system. The permissions granted to roles are taken into consideration during the authorization process for views, triggers, materialized query tables (MQTs), packages and SQL routines, unlike the permissions granted to groups. Permissions granted to groups are not considered during the authorization process for views, triggers, MQTs, packages and SQL routines, because the DB2 database system cannot discover when membership in a group changes, and so it cannot invalidate the objects, above, if appropriate.

Note: Permissions granted to roles that are granted to groups are not considered during the authorization process for views, triggers, MQTs, packages and SQL routines.

During an SQL statement processing, the permissions that the DB2 authorization model considers are the union of the following permissions:

1. The permissions granted to the primary authorization ID associated with the SQL statement
2. The permissions granted to the secondary authorization IDs (groups or roles) associated with the SQL statement
3. The permissions granted to PUBLIC, including roles that are granted to PUBLIC, directly or indirectly through other roles.
4. The permissions granted to the trusted context role, if applicable.

Authorization IDs in different contexts

An authorization ID is used for two purposes: identification and authorization checking. For example, the session authorization ID is used for initial authorization checking.

When referring to the use of an authorization ID in a specific context, the reference to the authorization is qualified to identify the context, as shown below.

Contextual reference to authorization ID
Definition

System authorization ID
The authorization ID used to do any initial authorization checking, such as checking for CONNECT privilege during CONNECT processing. As part of the authentication process during CONNECT processing, an authorization ID compatible with DB2 naming requirements is produced that represents the external user ID within the DB2 database system. The system authorization ID represents the user that created the connection. Use the SYSTEM_USER special register to see the current value of the system authorization ID. The system authorization ID cannot be changed for a connection.

Session authorization ID
The authorization ID used for any session authorization checking subsequent to the initial checks performed during CONNECT processing. The default value of the session authorization ID is the value of the system authorization ID. Use the SESSION_USER special register to see the current value of the session authorization ID. The USER special register is a synonym for the SESSION_USER special register. The session authorization ID can be changed by using the SET SESSION AUTHORIZATION statement.

Package authorization ID
The authorization ID used to bind a package to the database. This authorization ID is obtained from the value of the **OWNER** *authorization id* option of the BIND command. The package authorization ID is sometimes referred to as the package binder or package owner.

Routine owner authorization ID
The authorization ID listed in the system catalogs as the owner of the SQL routine that has been invoked.

Routine invoker authorization ID
The authorization ID that is the statement authorization ID for the statement that invoked an SQL routine.

Statement authorization ID
The authorization ID associated with a specific SQL statement that is to be used for any authorization requirements as well as for determining object ownership (where appropriate). It takes its value from the appropriate source authorization ID, depending on the type of SQL statement:

- Static SQL

  The package authorization ID is used.
- Dynamic SQL (from non-routine context)

  The table shows which authorization ID is used in each case:

Value of DYNAMICRULES option for issuing the package	**Authorization ID used**
RUN	Session authorization ID
BIND	Package authorization ID
DEFINERUN, INVOKERUN	Session authorization ID
DEFINEBIND, INVOKEBIND	Package authorization ID

- Dynamic SQL (from routine context)

  The table shows which authorization ID is used in each case:

Value of DYNAMICRULES option for issuing the package	**Authorization ID used**
DEFINERUN, DEFINEBIND	Routine owner authorization ID
INVOKERUN, INVOKEBIND	Routine invoker authorization ID

Use the CURRENT_USER special register to see the current value of the statement authorization ID. The statement authorization ID cannot be changed directly; it is changed automatically by the DB2 database system to reflect the nature of each SQL statement.

Authorization, privileges, and object ownership

Users (identified by an authorization ID) can successfully execute operations only if they have the authority to perform the specified function. To create a table, a user must be authorized to create tables; to alter a table, a user must be authorized to alter the table; and so forth.

The database manager requires that each user be specifically authorized to use each database function needed to perform a specific task. A user can acquire the necessary authorization through a grant of that authorization to their user ID or through membership in a role or a group that holds that authorization.

There are three forms of authorization, *administrative authority*, *privileges*, and *LBAC credentials*. In addition, ownership of objects brings with it a degree of authorization on the objects created. These forms of authorization are discussed below.

Administrative authority

The person or persons holding administrative authority are charged with the task of controlling the database manager and are responsible for the safety and integrity of the data.

System-level authorization

The system-level authorities provide varying degrees of control over instance-level functions:

- SYSADM (system administrator) authority

  The SYSADM (system administrator) authority provides control over all the resources created and maintained by the database manager. The system administrator possesses all the authorities of SYSCTRL, SYSMAINT, and SYSMON authority. The user who has SYSADM authority is responsible both for controlling the database manager, and for ensuring the safety and integrity of the data.
- SYSCTRL authority

  The SYSCTRL authority provides control over operations that affect system resources. For example, a user with SYSCTRL authority can create, update, start, stop, or drop a database. This user can also start or stop an instance, but cannot access table data. Users with SYSCTRL authority also have SYSMON authority.
- SYSMAINT authority

  The SYSMAINT authority provides the authority required to perform maintenance operations on all databases associated with an instance. A user with SYSMAINT authority can update the database configuration, backup a database or table space, restore an existing database, and monitor a database. Like SYSCTRL, SYSMAINT does not provide access to table data. Users with SYSMAINT authority also have SYSMON authority.
- SYSMON (system monitor) authority

  The SYSMON (system monitor) authority provides the authority required to use the database system monitor.

Database-level authorization

The database level authorities provide control within the database:

- DBADM (database administrator)

The DBADM authority level provides administrative authority over a single database. This database administrator possesses the privileges required to create objects and issue database commands.

The DBADM authority can only be granted by a user with SECADM authority. The DBADM authority cannot be granted to PUBLIC.

- SECADM (security administrator)

  The SECADM authority level provides administrative authority for security over a single database. The security administrator authority possesses the ability to manage database security objects (database roles, audit policies, trusted contexts, security label components, and security labels) and grant and revoke all database privileges and authorities. A user with SECADM authority can transfer the ownership of objects that they do not own. They can also use the AUDIT statement to associate an audit policy with a particular database or database object at the server.

  The SECADM authority has no inherent privilege to access data stored in tables. It can only be granted by a user with SECADM authority. The SECADM authority cannot be granted to PUBLIC.

- SQLADM (SQL administrator)

  The SQLADM authority level provides administrative authority to monitor and tune SQL statements within a single database. It can be granted by a user with ACCESSCTRL or SECADM authority.

- WLMADM (workload management administrator)

  The WLMADM authority provides administrative authority to manage workload management objects, such as service classes, work action sets, work class sets, and workloads. It can be granted by a user with ACCESSCTRL or SECADM authority.

- EXPLAIN (explain authority)

  The EXPLAIN authority level provides administrative authority to explain query plans without gaining access to data. It can only be granted by a user with ACCESSCTRL or SECADM authority.

- ACCESSCTRL (access control authority)

  The ACCESSCTRL authority level provides administrative authority to issue the following GRANT (and REVOKE) statements. ACCESSCTRL authority can only be granted by a user with SECADM authority. The ACCESSCTRL authority cannot be granted to PUBLIC.

  – GRANT (Database Authorities)

    ACCESSCTRL authority does not give the holder the ability to grant ACCESSCTRL, DATAACCESS, DBADM, or SECADM authority. Only a user who has SECADM authority can grant these authorities.

  – GRANT (Global Variable Privileges)
  – GRANT (Index Privileges)
  – GRANT (Module Privileges)
  – GRANT (Package Privileges)
  – GRANT (Routine Privileges)
  – GRANT (Schema Privileges)
  – GRANT (Sequence Privileges)
  – GRANT (Server Privileges)
  – GRANT (Table, View, or Nickname Privileges)
  – GRANT (Table Space Privileges)
  – GRANT (Workload Privileges)

  - GRANT (XSR Object Privileges)
- DATAACCESS (data access authority)

  The DATAACCESS authority level provides the following privileges and authorities. It can be granted only by a user who holds SECADM authority. The DATAACCESS authority cannot be granted to PUBLIC.
  - LOAD authority
  - SELECT, INSERT, UPDATE, DELETE privilege on tables, views, nicknames, and materialized query tables
  - EXECUTE privilege on packages
  - EXECUTE privilege on modules
  - EXECUTE privilege on routines

    Except on the audit routines: AUDIT_ARCHIVE, AUDIT_LIST_LOGS, AUDIT_DELIM_EXTRACT.
- Database authorities (non-administrative)

  To perform activities such as creating a table or a routine, or for loading data into a table, specific database authorities are required. For example, the LOAD database authority is required for use of the load utility to load data into tables (a user must also have INSERT privilege on the table).

Privileges

A privilege is a permission to perform an action or a task. Authorized users can create objects, have access to objects they own, and can pass on privileges on their own objects to other users by using the GRANT statement.

Privileges may be granted to individual users, to groups, or to PUBLIC. PUBLIC is a special group that consists of all users, including future users. Users that are members of a group will indirectly take advantage of the privileges granted to the group, where groups are supported.

The CONTROL privilege: Possessing the CONTROL privilege on an object allows a user to access that database object, and to grant and revoke privileges to or from other users on that object.

Note: The CONTROL privilege only apples to tables, views, nicknames, indexes, and packages.

If a different user requires the CONTROL privilege to that object, a user with SECADM or ACCESSCTRL authority could grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked from the object owner, however, the object owner can be changed by using the TRANSFER OWNERSHIP statement.

Individual privileges: Individual privileges can be granted to allow a user to carry out specific tasks on specific objects. Users with the administrative authorities ACCESSCTRL or SECADM, or with the CONTROL privilege, can grant and revoke privileges to and from users.

Individual privileges and database authorities allow a specific function, but do not include the right to grant the same privileges or authorities to other users. The right to grant table, view, schema, package, routine, and sequence privileges to others can be extended to other users through the WITH GRANT OPTION on the GRANT statement. However, the WITH GRANT OPTION does not allow the

person granting the privilege to revoke the privilege once granted. You must have SECADM authority, ACCESSCTRL authority, or the CONTROL privilege to revoke the privilege.

Privileges on objects in a package or routine: When a user has the privilege to execute a package or routine, they do not necessarily require specific privileges on the objects used in the package or routine. If the package or routine contains static SQL or XQuery statements, the privileges of the owner of the package are used for those statements. If the package or routine contains dynamic SQL or XQuery statements, the authorization ID used for privilege checking depends on the setting of the **DYNAMICRULES** BIND option of the package issuing the dynamic query statements, and whether those statements are issued when the package is being used in the context of a routine.

A user or group can be authorized for any combination of individual privileges or authorities. When a privilege is associated with an object, that object must exist. For example, a user cannot be given the SELECT privilege on a table unless that table has previously been created.

Note: Care must be taken when an authorization name representing a user or group is granted authorities and privileges in the following situation:

- A user or group with that name has not been created
- A user or group with that name has been deleted

At some later time, a user or group can be created with that name and automatically receive all of the authorities and privileges associated with that authorization name. For a user or group that is deleted, authorities and privileges to the authorization name representing the user or group should be revoked.

The REVOKE statement is used to revoke previously granted privileges. The revoking of a privilege from an authorization name revokes the privilege granted by all authorization names.

Revoking a privilege from an authorization name does not revoke that same privilege from any other authorization names that were granted the privilege by that authorization name. For example, assume that CLAIRE grants SELECT WITH GRANT OPTION to RICK, then RICK grants SELECT to BOBBY and CHRIS. If CLAIRE revokes the SELECT privilege from RICK, BOBBY and CHRIS still retain the SELECT privilege.

LBAC credentials

Label-based access control (LBAC) lets the security administrator decide exactly who has write access and who has read access to individual rows and individual columns. The security administrator configures the LBAC system by creating security policies. A security policy describes the criteria used to decide who has access to what data. Only one security policy can be used to protect any one table but different tables can be protected by different security policies.

After creating a security policy, the security administrator creates database objects, called security labels and exemptions that are part of that policy. A security label describes a certain set of security criteria. An exemption allows a rule for comparing security labels not to be enforced for the user who holds the exemption, when they access data protected by that security policy.

Once created, a security label can be associated with individual columns and rows in a table to protect the data held there. Data that is protected by a security label is called protected data. A security administrator allows users access to protected data by granting them security labels. When a user tries to access protected data, that user's security label is compared to the security label protecting the data. The protecting label blocks some security labels and does not block others.

Object ownership

When an object is created, one authorization ID is assigned *ownership* of the object. Ownership means the user is authorized to reference the object in any applicable SQL or XQuery statement.

When an object is created within a schema, the authorization ID of the statement must have the required privilege to create objects in the implicitly or explicitly specified schema. That is, the authorization name must either be the owner of the schema, or possess the CREATEIN privilege on the schema.

Note: This requirement is not applicable when creating table spaces, buffer pools or database partition groups. These objects are not created in schemas.

When an object is created, the authorization ID of the statement is the definer of that object and by default becomes the owner of the object after it is created.

Note: One exception exists. If the AUTHORIZATION option is specified for the CREATE SCHEMA statement, any other object that is created as part of the CREATE SCHEMA operation is owned by the authorization ID specified by the AUTHORIZATION option. Any objects that are created in the schema after the initial CREATE SCHEMA operation, however, are owned by the authorization ID associated with the specific CREATE statement.

For example, the statement `CREATE SCHEMA SCOTTSTUFF AUTHORIZATION SCOTT CREATE TABLE T1 (C1 INT)` creates the schema `SCOTTSTUFF` and the table `SCOTTSTUFF.T1`, which are both owned by SCOTT. Assume that the user BOBBY is granted the CREATEIN privilege on the `SCOTTSTUFF` schema and creates an index on the `SCOTTSTUFF.T1` table. Because the index is created after the schema, BOBBY owns the index on `SCOTTSTUFF.T1`.

Privileges are assigned to the object owner based on the type of object being created:

- The CONTROL privilege is implicitly granted on newly created tables, indexes, and packages. This privilege allows the object creator to access the database object, and to grant and revoke privileges to or from other users on that object. If a different user requires the CONTROL privilege to that object, a user with ACCESSCTRL or SECADM authority must grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked by the object owner.
- The CONTROL privilege is implicitly granted on newly created views if the object owner has the CONTROL privilege on all the tables, views, and nicknames referenced by the view definition.
- Other objects like triggers, routines, sequences, table spaces, and buffer pools do not have a CONTROL privilege associated with them. The object owner does, however, automatically receive each of the privileges associated with the object and those privileges are with the WITH GRANT OPTION, where supported. Therefore the object owner can provide these privileges to other users by using the GRANT statement. For example, if USER1 creates a table space, USER1

automatically has the USEAUTH privilege with the WITH GRANT OPTION on this table space and can grant the USEAUTH privilege to other users. In addition, the object owner can alter, add a comment on, or drop the object. These authorizations are implicit for the object owner and cannot be revoked.

Certain privileges on the object, such as altering a table, can be granted by the owner, and can be revoked from the owner by a user who has ACCESSCTRL or SECADM authority. Certain privileges on the object, such as commenting on a table, cannot be granted by the owner and cannot be revoked from the owner. Use the TRANSFER OWNERSHIP statement to move these privileges to another user. When an object is created, the authorization ID of the statement is the definer of that object and by default becomes the owner of the object after it is created. However, when you use the BIND command to create a package and you specify the **OWNER** *authorization id* option, the owner of objects created by the static SQL statements in the package is the value of *authorization id*. In addition, if the AUTHORIZATION clause is specified on a CREATE SCHEMA statement, the authorization name specified after the AUTHORIZATION keyword is the owner of the schema.

A security administrator or the object owner can use the TRANSFER OWNERSHIP statement to change the ownership of a database object. An administrator can therefore create an object on behalf of an authorization ID, by creating the object using the authorization ID as the qualifier, and then using the TRANSFER OWNERSHIP statement to transfer the ownership that the administrator has on the object to the authorization ID.

Object creation, ownership, and privileges

When an object is created, one authorization name is assigned *ownership* of the object. Ownership means that the user is authorized to reference the object in any SQL or XQuery statement.

When an object is created within a schema, the authorization ID of the statement must have the required privilege to create objects in the implicitly or explicitly specified schema. That is, the authorization name must either be the owner of the schema, or possess the CREATEIN privilege on the schema.

Note: This requirement is not applicable when creating table spaces, buffer pools or database partition groups. These objects are not created in schemas.

When an object is created, the authorization ID of the statement is the owner of that object.

Note: One exception exists. If the AUTHORIZATION option is specified for the CREATE SCHEMA statement, any other object that is created as part of the CREATE SCHEMA operation is owned by the authorization ID specified by the AUTHORIZATION option. Any objects that are created in the schema after the initial CREATE SCHEMA operation, however, are owned by the authorization ID associated with the specific CREATE statement.

For example, the statement

```
CREATE SCHEMA SCOTTSTUFF AUTHORIZATION SCOTT CREATE TABLE T1 (C! INT)
```

creates the schema `SCOTTSTUFF` and the table `SCOTTSTUFF.T1`, which are both owned by SCOTT. Assume that the user BOBBY is granted the CREATEIN privilege on the

SCOTTSTUFF schema and creates an index on the SCOTTSTUFF.T1 table. Because the index is created after the schema, BOBBY owns the index on SCOTTSTUFF.T1.

Privileges are assigned to the object owner based on the type of object being created:

- The CONTROL privilege is implicitly granted on newly created tables, indexes, and packages. This privilege allows the object creator to access the database object, and to grant and revoke privileges to or from other users on that object. If a different user requires the CONTROL privilege to that object, a user with SYSADM or DBADM authority must grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked by the object owner.
- The CONTROL privilege is implicitly granted on newly created views if the object owner has the CONTROL privilege on all the tables, views, and nicknames referenced by the view definition.
- Other objects like triggers, routines, sequences, table spaces, and buffer pools do not have a CONTROL privilege associated with them. The object owner does, however, automatically receive each of the privileges associated with the object (and can provide these privileges to other users, where supported, by using the WITH GRANT option of the GRANT statement). In addition, the object owner can alter, add a comment on, or drop the object. These authorizations are implicit for the object owner and cannot be revoked.

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.

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, copy, and drop objects in the schema. This 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. A user with ACCESSCTRL or SECADM authority can change the privileges that are held by users on any schema. Therefore, access to create, alter, copy, and drop objects in any schema (even one that was implicitly created) can be controlled.

Details on privileges, authorities, and authorization

Each authority is discussed in this section followed by the different privileges.

Default privileges granted on creating a database

When you create a database, default database level authorities and default object level privileges are granted to you within that database.

The authorities and privileges that you are granted are listed according to the system catalog views where they are recorded:

1. SYSCAT.DBAUTH
   - The database creator is granted the following authorities:
     - ACCESSCTRL
     - DATAACCESS
     - DBADM
     - SECADM
   - In a non-restrictive database, the special group PUBLIC is granted the following authorities:

- CREATETAB
- BINDADD
- CONNECT
- IMPLICIT_SCHEMA

2. SYSCAT.TABAUTH

   In a non-restrictive database, the special group PUBLIC is granted the following privileges:

   - SELECT on all SYSCAT and SYSIBM tables
   - SELECT and UPDATE on all SYSSTAT tables
   - SELECT on the following views in schema SYSIBMADM:
     - ALL_*
     - USER_*
     - ROLE_*
     - SESSION_*
     - DICTIONARY
     - TAB

3. SYSCAT.ROUTINEAUTH

   In a non-restrictive database, the special group PUBLIC is granted the following privileges:

   - EXECUTE with GRANT on all procedures in schema SQLJ
   - EXECUTE with GRANT on all functions and procedures in schema SYSFUN
   - EXECUTE with GRANT on all functions and procedures in schema SYSPROC (except audit routines)
   - EXECUTE on all table functions in schema SYSIBM
   - EXECUTE on all other procedures in schema SYSIBM
   - EXECUTE on the following modules in schema SYSIBMADM:
     - DBMS_JOB
     - DBMS_LOB
     - DBMS_OUTPUT
     - DBMS_SQL
     - DBMS_UTILITY

4. SYSCAT.PACKAGEAUTH
   - The database creator is granted the following privileges:
     - CONTROL on all packages created in the NULLID schema
     - BIND with GRANT on all packages created in the NULLID schema
     - EXECUTE with GRANT on all packages created in the NULLID schema
   - In a non-restrictive database, the special group PUBLIC is granted the following privileges:
     - BIND on all packages created in the NULLID schema
     - EXECUTE on all packages created in the NULLID schema

5. SYSCAT.SCHEMAAUTH

   In a non-restrictive database, the special group PUBLIC is granted the following privileges:

   - CREATEIN on schema SQLJ
   - CREATEIN on schema NULLID

6. SYSCAT.TBSPACEAUTH

In a non-restrictive database, the special group PUBLIC is granted the USE privilege on table space USERSPACE1.

7. SYSCAT.WORKLOADAUTH

   In a non-restrictive database, the special group PUBLIC is granted the USAGE privilege on SYSDEFAULTUSERWORKLOAD.

A non-restrictive database is a database created without the RESTRICTIVE option on the CREATE DATABASE command.

System administration authority (SYSADM)

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 assigned to the group specified by the **sysadm_group** configuration parameter. Membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSADM authority can perform the following functions:

- Upgrade a database
- Restore a database
- Change the database manager configuration file (including specifying the groups having SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority)

A user with SYSADM authority can grant and revoke table space privileges and can also use any table space.

Note: When a user with SYSADM authority creates a database, that user is automatically granted ACCESSCTRL, DATAACCESS, DBADM and SECADM authority on the database. If you want to prevent that user from accessing that database as a database administrator or a security administrator, you must explicitly revoke these database authorities from the user.

In releases prior to Version 9.7, SYSADM authority included implicit DBADM authority and also provided the ability to grant and revoke all authorities and privileges. In Version 9.7, the DB2 authorization model has been updated to clearly separate the duties of the system administrator, the database administrator, and the security administrator. As part of this enhancement, the abilities given by the SYSADM authority have been reduced.

In Version 9.7, only SECADM authority provides the ability to grant and revoke all authorities and privileges.

For a user holding SYSADM authority to obtain the same capabilities as in Version 9.5 (other than the ability to grant SECADM authority), the security administrator must explicitly grant the user DBADM authority and grant the user the new DATAACCESS and ACCESSCTRL authorities. These new authorities can be granted by using the GRANT DBADM ON DATABASE statement with the WITH DATAACCESS and WITH ACCESSCTRL options of that statement, which are default options. The DATAACCESS authority is the authority that allows access to data within a specific database, and the ACCESSCTRL authority is the authority that allows a user to grant and revoke privileges and non-administrative authorities within a specific database.

Considerations for the Windows LocalSystem account

On Windows systems, when the **sysadm_group** database manager configuration parameter is not specified, the LocalSystem account is considered a system administrator (holding SYSADM authority). Any DB2 application that is run by LocalSystem is affected by the change in scope of SYSADM authority in Version 9.7. These applications are typically written in the form of Windows services and run under the LocalSystem account as the service logon account. If there is a need for these applications to perform database actions that are no longer within the scope of SYSADM, you must grant the LocalSystem account the required database privileges or authorities. For example, if an application requires database administrator capabilities, grant the LocalSystem account DBADM authority using the GRANT (Database Authorities) statement. Note that the authorization ID for the LocalSystem account is SYSTEM.

System control authority (SYSCTRL)

SYSCTRL authority is the highest level of system control authority. This authority provides the ability to perform maintenance and utility operations against the database manager instance and its databases. These operations can affect system resources, but they do not allow direct access to data in the databases.

System control authority is designed for users administering a database manager instance containing sensitive data.

SYSCTRL authority is assigned to the group specified by the **sysctrl_group** configuration parameter. If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSCTRL authority or higher can do the following:

- Update a database, node, or distributed connection services (DCS) directory
- Force users off the system
- Create or drop a database
- Drop, create, or alter a table space
- Use any table space
- Restore to a new or an existing database.

In addition, a user with SYSCTRL authority can perform the functions of users with system maintenance authority (SYSMAINT) and system monitor authority (SYSMON).

Users with SYSCTRL authority also have the implicit privilege to connect to a database.

Note: When users with SYSCTRL authority create databases, they are automatically granted explicit ACCESSCTRL, DATAACCESS, DBADM, and SECADM authorities on the database. If the database creator is removed from the SYSCTRL group, and if you want to also prevent them from accessing that database as an administrator, you must explicitly revoke the four administrative authorities, above.

System maintenance authority (SYSMAINT)

SYSMAINT authority is the second level of system control authority. This authority provides the ability to perform maintenance and utility operations against the database manager instance and its databases. These operations can affect system resources, but they do not allow direct access to data in the databases.

System maintenance authority is designed for users maintaining databases within a database manager instance that contains sensitive data.

SYSMAINT authority is assigned to the group specified by the **sysmaint_group** configuration parameter. If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSMAINT or higher system authority can do the following:

- Back up a database or table space
- Restore to an existing database
- Perform roll forward recovery
- Start or stop an instance
- Restore a table space
- Run a trace, using the db2trc command
- Take database system monitor snapshots of a database manager instance or its databases.

A user with SYSMAINT authority can do the following:

- Query the state of a table space
- Update log history files
- Quiesce a table space
- Reorganize a table
- Collect catalog statistics using the RUNSTATS utility.

Users with SYSMAINT authority also have the implicit privilege to connect to a database, and can perform the functions of users with system monitor authority (SYSMON).

System monitor authority (SYSMON)

SYSMON authority provides the ability to take database system monitor snapshots of a database manager instance or its databases.

SYSMON authority is assigned to the group specified by the **sysmon_group** configuration parameter. If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

SYSMON authority enables the user to run the following commands:

- GET DATABASE MANAGER MONITOR SWITCHES
- GET MONITOR SWITCHES
- GET SNAPSHOT
- LIST (some commands):
  - LIST ACTIVE DATABASES
  - LIST APPLICATIONS
  - LIST DATABASE PARTITION GROUPS

  - LIST DCS APPLICATIONS
  - LIST PACKAGES
  - LIST TABLES
  - LIST TABLESPACE CONTAINERS
  - LIST TABLESPACES
  - LIST UTILITIES
- RESET MONITOR
- UPDATE MONITOR SWITCHES

SYSMON authority enables the user to use the following APIs:
- db2GetSnapshot - Get Snapshot
- db2GetSnapshotSize - Estimate Size Required for db2GetSnapshot() Output Buffer
- db2MonitorSwitches - Get/Update Monitor Switches
- db2mtrk - Memory tracker
- db2ResetMonitor - Reset Monitor

SYSMON authority enables the user use the following SQL table functions:
- All snapshot table functions without previously running SYSPROC.SNAP_WRITE_FILE

  SYSPROC.SNAP_WRITE_FILE takes a snapshot and saves its content into a file. If any snapshot table functions are called with null input parameters, the file content is returned instead of a real-time system snapshot.

Database authorities

Each database authority allows the authorization ID holding it to perform some particular type of action on the database as a whole. Database authorities are different from privileges, which allow a certain action to be taken on a particular database object, such as a table or an index.

These are the database authorities.

ACCESSCTRL
Allows the holder to grant and revoke all object privileges and database authorities except for privileges on the audit routines, and ACCESSCTRL, DATAACCESS, DBADM, and SECADM authority.

BINDADD
Allows the holder to create new packages in the database.

CONNECT
Allows the holder to connect to the database.

CREATETAB
Allows the holder to create new tables in the database.

CREATE_EXTERNAL_ROUTINE
Allows the holder to create a procedure for use by applications and other users of the database.

CREATE_NOT_FENCED_ROUTINE
Allows the holder to create a user-defined function (UDF) or procedure that is *not fenced*. CREATE_EXTERNAL_ROUTINE is automatically granted to any user who is granted CREATE_NOT_FENCED_ROUTINE.

Attention: The database manager does not protect its storage or control blocks from UDFs or procedures that are not fenced. A user with this authority must, therefore, be very careful to test their UDF extremely well before registering it as not fenced.

DATAACCESS
Allows the holder to access data stored in database tables.

DBADM
Allows the holder to act as the database administrator. In particular it gives the holder all of the other database authorities except for ACCESSCTRL, DATAACCESS, and SECADM.

EXPLAIN
Allows the holder to explain query plans without requiring them to hold the privileges to access data in the tables referenced by those query plans.

IMPLICIT_SCHEMA
Allows any user to create a schema implicitly by creating an object using a CREATE statement with a schema name that does not already exist. SYSIBM becomes the owner of the implicitly created schema and PUBLIC is given the privilege to create objects in this schema.

LOAD
Allows the holder to load data into a table

QUIESCE_CONNECT
Allows the holder to access the database while it is quiesced.

SECADM
Allows the holder to act as a security administrator for the database.

SQLADM
Allows the holder to monitor and tune SQL statements.

WLMADM
Allows the holder to act as a workload administrator. In particular, the holder of WLMADM authority can create and drop workload manager objects, grant and revoke workload manager privileges, and execute workload manager routines.

Only authorization IDs with the SECADM authority can grant the ACCESSCTRL, DATAACCESS, DBADM, and SECADM authorities. All other authorities can be granted by authorization IDs that hold ACCESSCTRL or SECADM authorities.

To remove any database authority from PUBLIC, an authorization ID with ACCESSCTRL or SECADM authority must explicitly revoke it.

Security administration authority (SECADM)

SECADM authority is the security administration authority for a specific database. This authority allows you to create and manage security-related database objects and to grant and revoke all database authorities and privileges. Additionally, the security administrator can execute, and manage who else can execute, the audit system routines.

SECADM authority has the ability to SELECT from the catalog tables and catalog views, but cannot access data stored in user tables.

SECADM authority can be granted only by the security administrator (who holds SECADM authority) and can be granted to a user, a group, or a role. PUBLIC cannot obtain the SECADM authority directly or indirectly.

SECADM authority gives a user the ability to perform the following operations:

- Create, alter, comment on, and drop:
  - Audit policies
  - Security label components
  - Security policies
  - Trusted contexts
- Create, comment on, and drop:
  - Roles
  - Security labels
- Grant and revoke database privileges and authorities
- Execute the following audit routines to perform the specified tasks:
  - The SYSPROC.AUDIT_ARCHIVE stored procedure and table function archive audit logs.
  - The SYSPROC.AUDIT_LIST_LOGS table function allows you to locate logs of interest.
  - The SYSPROC.AUDIT_DELIM_EXTRACT stored procedure extracts data into delimited files for analysis.

  Also, the security administrator can grant and revoke EXECUTE privilege on these routines, therefore enabling the security administrator to delegate these tasks, if desired. Only the security administrator can grant EXECUTE privilege on these routines. EXECUTE privilege WITH GRANT OPTION cannot be granted for these routines (SQLSTATE 42501).
- Use of the AUDIT statement to associate an audit policy with a particular database or database object at the server
- Use of the TRANSFER OWNERSHIP statement to transfer objects not owned by the authorization ID of the statement

No other authority gives these abilities.

The instance owner does not have SECADM authority by default.

Only the security administrator has the ability to grant other users, groups, or roles the ACCESSCTRL, DATAACCESS, DBADM, and SECADM authorities.

In Version 9.7, the DB2 authorization model has been updated to clearly separate the duties of the system administrator, the database administrator, and the security administrator. As part of this enhancement, the abilities given by the SECADM authority have been extended. In releases prior to Version 9.7, SECADM authority did not provide the ability to grant and revoke all privileges and authorities. Also, SECADM authority could be granted only to a user, not to a role or a group. Additionally, SECADM authority did not provide the ability to grant EXECUTE privilege to other users on the audit system-defined procedures and table function.

Database administration authority (DBADM)

DBADM authority is an administrative authority for a specific database. The database administrator possesses the privileges required to create objects and issue database commands. In addition, users with DBADM authority have SELECT

privilege on the system catalog tables and views, and can execute all system-defined DB2 routines, except audit routines.

DBADM authority can only be granted or revoked by the security administrator (who holds SECADM authority) and can be granted to a user, a group, or a role. PUBLIC cannot obtain the DBADM authority either directly or indirectly.

Holding the DBADM authority for a database allows a user to perform these actions on that database:

- Create, alter, and drop non-security related database objects
- Read log files
- Create, activate, and drop event monitors
- Query the state of a table space
- Update log history files
- Quiesce a table space
- Reorganize a table
- Collect catalog statistics using the RUNSTATS utility

SQLADM authority and WLMADM authority are subsets of the DBADM authority. WLMADM authority has the additional ability to grant the USAGE privilege on workloads.

Granting DATAACCESS authority with DBADM authority

The security administrator can specify whether a database administrator can access data within the database. DATAACCESS authority is the authority that allows access to data within a specific database. The security administrator can use the WITH DATAACCESS option of the GRANT DBADM ON DATABASE statement to provide a database administrator with this ability. If neither the WITH DATAACCCESS or WITHOUT DATAACCCESS options are specified, by default DATAACCESS authority is granted.

To grant database administrator authority without DATAACCESS authority, use GRANT DBADM WITHOUT DATAACCESS in your SQL statement.

Granting ACCESSCTRL authority with DBADM authority

The security administrator can specify whether a database administrator can grant and revoke privileges within the database. ACCESSCTRL authority is the authority that allows a user to grant and revoke privileges and non-administrative authorities within a specific database. The security administrator can use the WITH ACCESSCTRL option of the GRANT DBADM ON DATABASE statement to provide a database administrator with this ability. If neither the WITH ACCCESSCTRL or WITHOUT ACCCESSCTRL options are specified, by default ACCESSCTRL authority is granted.

To grant database administrator authority without ACCESSCTRL authority, use GRANT DBADM WITHOUT ACCESSCTRL in your SQL statement.

Revoking DBADM authority

If a security administrator has granted DBADM authority that includes DATAACCESS or ACCESSCTRL authority, to revoke these authorities, the security

administrator must explicitly revoke DATAACCESS or ACCESSCTRL authority. For example, if the security administrator grants DBADM authority to a user:

```
GRANT DBADM ON DATABASE TO user1
```

By default, DATAACCESS and ACCESSCTRL authority are also granted to user1.

Later, the security administrator revokes DBADM authority from user1:

```
REVOKE DBADM ON DATABASE FROM user1
```

Now user1 no longer holds DBADM authority, but still has both DATAACCESS and ACCESSCTRL authority.

To revoke these remaining authorities, the security administrator needs to revoke them explicitly:

```
REVOKE ACCESSCTRL, DATAACCESS ON DATABASE FROM user1
```

Differences for DBADM authority in prior releases

In Version 9.7, the DB2 authorization model has been updated to clearly separate the duties of the system administrator, the database administrator, and the security administrator. As part of this enhancement, the abilities given by the DBADM authority have changed. In releases prior to Version 9.7, DBADM authority automatically included the ability to access data and to grant and revoke privileges for a database. In Version 9.7, these abilities are given by the new authorities, DATAACCESS and ACCESSCTRL, respectively, as explained earlier.

Also, in releases prior to Version 9.7, granting DBADM authority automatically granted the following authorities too:

- BINDADD
- CONNECT
- CREATETAB
- CREATE_EXTERNAL_ROUTINE
- CREATE_NOT_FENCED_ROUTINE
- IMPLICIT_SCHEMA
- QUIESCE_CONNECT
- LOAD

Prior to Version 9.7, when DBADM authority was revoked these authorities were not revoked.

In Version 9.7, these authorities are now part of DBADM authority. When DBADM authority is revoked in Version 9.7, these authorities are lost.

However, if a user held DBADM authority when you upgraded to Version 9.7, these authorities are not lost if DBADM authority is revoked. Revoking DBADM authority in Version 9.7 causes a user to lose these authorities only if they acquired them through holding DBADM authority that was granted in Version 9.7.

LOAD authority

Users having LOAD authority at the database level, as well as INSERT privilege on a table, can use the LOAD command to load data into a table.

Note: Having DATAACCESS authority gives a user full access to the LOAD command.

Users having LOAD authority at the database level, as well as INSERT privilege on a table, can LOAD RESTART or LOAD TERMINATE if the previous load operation is a load to insert data.

Users having LOAD authority at the database level, as well as the INSERT and DELETE privileges on a table, can use the LOAD REPLACE command.

If the previous load operation was a load replace, the DELETE privilege must also have been granted to that user before the user can LOAD RESTART or LOAD TERMINATE.

If the exception tables are used as part of a load operation, the user must have INSERT privilege on the exception tables.

The user with this authority can perform QUIESCE TABLESPACES FOR TABLE, RUNSTATS, and LIST TABLESPACES commands.

Implicit schema authority (IMPLICIT_SCHEMA) considerations

When a new database is created, PUBLIC is given IMPLICIT_SCHEMA database authority, unless the **RESTRICTIVE** option is specified on the CREATE DATABASE command.

With the IMPLICIT_SCHEMA authority, a user can create a schema by creating an object and specifying a schema name that does not already exist. SYSIBM becomes the owner of the implicitly created schema and PUBLIC is given the privilege to create objects in this schema.

If control of who can implicitly create schema objects is required for the database, IMPLICIT_SCHEMA database authority should be revoked from PUBLIC. Once this is done, there are only three (3) ways that a schema object is created:

- Any user can create a schema using their own authorization name on a CREATE SCHEMA statement.
- Any user with DBADM authority can explicitly create any schema which does not already exist, and can optionally specify another user as the owner of the schema.
- Any user with DBADM authority has IMPLICIT_SCHEMA database authority, so that they can implicitly create a schema with any name at the time they are creating other database objects. SYSIBM becomes the owner of the implicitly created schema and PUBLIC has the privilege to create objects in the schema.

Schema privileges

Schema privileges are in the object privilege category.

Object privileges are shown in Figure 18 on page 99.

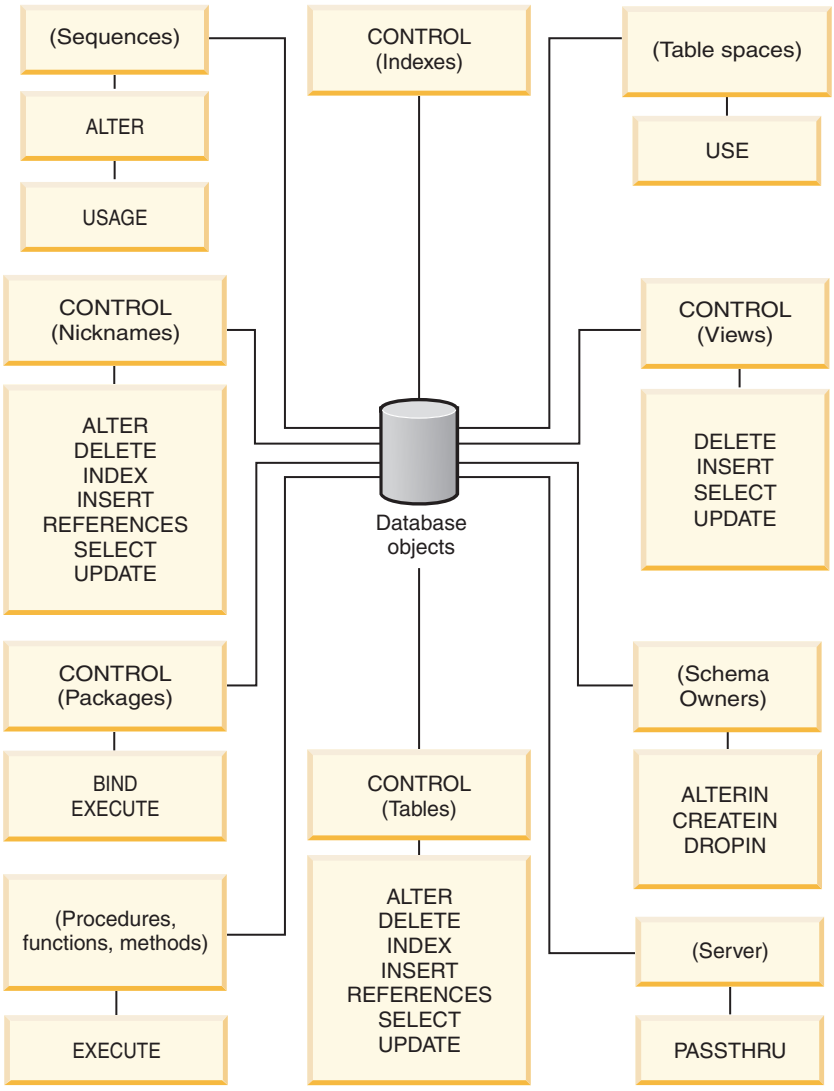


Figure 18. Object Privileges

Schema privileges involve actions on schemas in a database. A user, group, role, or PUBLIC can be granted any of the following privileges:

- CREATEIN allows the user to create objects within the schema.
- ALTERIN allows the user to alter objects within the schema.
- DROPIN allows the user to drop objects from within the schema.

The owner of the schema has all of these privileges and the ability to grant them to others. The objects that are manipulated within the schema object include: tables, views, indexes, packages, data types, functions, triggers, procedures, and aliases.

Table space privileges

The table space privileges involve actions on the table spaces in a database. A user can be granted the USE privilege for a table space, which then allows them to create tables within the table space.

The owner of the table space is granted USE privilege with the WITH GRANT OPTION on the table space when it is created. Also, users who hold SECADM or ACCESSCTRL authority have the ability to grant USE privilege on the table space.

Users who hold SYSADM or SYSCTRL authority are able to use any table space.

By default, at database creation time the USE privilege for table space USERSPACE1 is granted to PUBLIC, although this privilege can be revoked.

The USE privilege cannot be used with SYSCATSPACE or any system temporary table spaces.

Table and view privileges

Table and view privileges involve actions on tables or views in a database.

A user must have CONNECT authority on the database to use any of the following privileges:

- CONTROL provides the user with all privileges for a table or view including the ability to drop it, and to grant and revoke individual table privileges. You must have ACCESSCTRL or SECADM authority to grant CONTROL. The creator of a table automatically receives CONTROL privilege on the table. The creator of a view automatically receives CONTROL privilege only if they have CONTROL privilege on all tables, views, and nicknames referenced in the view definition.
- ALTER allows the user to modify on a table, for example, to add columns or a unique constraint to the table. A user with ALTER privilege can also COMMENT ON a table, or on columns of the table. For information about the possible modifications that can be performed on a table, see the ALTER TABLE and COMMENT statements.
- DELETE allows the user to delete rows from a table or view.
- INDEX allows the user to create an index on a table. Creators of indexes automatically have CONTROL privilege on the index.
- INSERT allows the user to insert a row into a table or view, and to run the IMPORT utility.
- REFERENCES allows the user to create and drop a foreign key, specifying the table as the parent in a relationship. The user might have this privilege only on specific columns.
- SELECT allows the user to retrieve rows from a table or view, to create a view on a table, and to run the EXPORT utility.
- UPDATE allows the user to change an entry in a table, a view, or for one or more specific columns in a table or view. The user may have this privilege only on specific columns.

The privilege to grant these privileges to others may also be granted using the WITH GRANT OPTION on the GRANT statement.

Note: When a user or group is granted CONTROL privilege on a table, all other privileges on that table are automatically granted WITH GRANT OPTION. If you subsequently revoke the CONTROL privilege on the table from a user, that user will still retain the other privileges that were automatically granted. To revoke all the privileges that are granted with the CONTROL privilege, you must either explicitly revoke each individual privilege or specify the ALL keyword on the REVOKE statement, for example:

```
REVOKE ALL
  ON EMPLOYEE FROM USER HERON
```

When working with typed tables, there are implications regarding table and view privileges.

Note: Privileges may be granted independently at every level of a table hierarchy. As a result, a user granted a privilege on a supertable within a hierarchy of typed tables may also indirectly affect any subtables. However, a user can only operate directly on a subtable if the necessary privilege is held on that subtable.

The supertable/subtable relationships among the tables in a table hierarchy mean that operations such as SELECT, UPDATE, and DELETE will affect the rows of the operation's target table and all its subtables (if any). This behavior can be called *substitutability*. For example, suppose that you have created an Employee table of type Employee_t with a subtable Manager of type Manager_t. A manager is a (specialized) kind of employee, as indicated by the type/subtype relationship between the structured types Employee_t and Manager_t and the corresponding table/subtable relationship between the tables Employee and Manager. As a result of this relationship, the SQL query:

```
SELECT * FROM Employee
```

will return the object identifier and Employee_t attributes for both employees and managers. Similarly, the update operation:

```
UPDATE Employee SET Salary = Salary + 1000
```

will give a thousand dollar raise to managers as well as regular employees.

A user with SELECT privilege on Employee will be able to perform this SELECT operation even if they do not have an explicit SELECT privilege on Manager. However, such a user will not be permitted to perform a SELECT operation directly on the Manager subtable, and will therefore not be able to access any of the non-inherited columns of the Manager table.

Similarly, a user with UPDATE privilege on Employee will be able to perform an UPDATE operation on Manager, thereby affecting both regular employees and managers, even without having the explicit UPDATE privilege on the Manager table. However, such a user will not be permitted to perform UPDATE operations directly on the Manager subtable, and will therefore not be able to update non-inherited columns of the Manager table.

Package privileges

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. Package privileges enable a user to create and manipulate packages.

The user must have CONNECT authority on the database to use any of the following privileges:

- CONTROL provides the user with the ability to rebind, drop, or execute a package as well as the ability to extend those privileges to others. The creator of a package automatically receives this privilege. A user with CONTROL privilege is granted the BIND and EXECUTE privileges, and can also grant these privileges to other users by using the GRANT statement. (If a privilege is granted using WITH GRANT OPTION, a user who receives the BIND or EXECUTE privilege can, in turn, grant this privilege to other users.) To grant CONTROL privilege, the user must have ACCESSCTRL or SECADM authority.
- BIND privilege on a package allows the user to rebind or bind that package and to add new package versions of the same package name and creator.
- EXECUTE allows the user to execute or run a package.

Note: All package privileges apply to all VERSIONs that share the same package name and creator.

In addition to these package privileges, the BINDADD database authority allows users to create new packages or rebind an existing package in the database.

Objects referenced by nicknames need to pass authentication checks at the data sources containing the objects. In addition, package users must have the appropriate privileges or authority levels for data source objects at the data source.

It is possible that packages containing nicknames might require additional authorization steps because DB2 database uses dynamic queries when communicating with DB2 Family data sources. The authorization ID running the package at the data source must have the appropriate authority to execute the package dynamically at that data source.

Index privileges

The creator of an index or an index specification automatically receives CONTROL privilege on the index. CONTROL privilege on an index is really the ability to drop the index. To grant CONTROL privilege on an index, a user must have ACCESSCTRL or SECADM authority.

The table-level INDEX privilege allows a user to create an index on that table.

The nickname-level INDEX privilege allows a user to create an index specification on that nickname.

Sequence privileges

The creator of a sequence automatically receives the USAGE and ALTER privileges on the sequence. The USAGE privilege is needed to use NEXT VALUE and PREVIOUS VALUE expressions for the sequence.

To allow other users to use the NEXT VALUE and PREVIOUS VALUE expressions, sequence privileges must be granted to PUBLIC. This allows all users to use the expressions with the specified sequence.

ALTER privilege on the sequence allows the user to perform tasks such as restarting the sequence or changing the increment for future sequence values. The creator of the sequence can grant the ALTER privilege to other users, and if WITH GRANT OPTION is used, these users can, in turn, grant these privileges to other users.

Routine privileges

Execute privileges involve actions on all types of routines such as functions, procedures, and methods within a database. Once having EXECUTE privilege, a user can then invoke that routine, create a function that is sourced from that routine (applies to functions only), and reference the routine in any DDL statement such as CREATE VIEW or CREATE TRIGGER.

The user who defines the externally stored procedure, function, or method receives EXECUTE WITH GRANT privilege. If the EXECUTE privilege is granted to another user via WITH GRANT OPTION, that user can, in turn, grant the EXECUTE privilege to another user.

Authorizations and binding of routines that contain SQL

When discussing routine level authorization it is important to define some roles related to routines, the determination of the roles, and the privileges related to these roles:

Package Owner

The owner of a particular package that participates in the implementation of a routine. The package owner is the user who executes the BIND command to bind a package with a database, unless the OWNER precompile/BIND option is used to override the package ownership and set it to an alternate user. Upon execution of the BIND command, the package owner is granted EXECUTE WITH GRANT privilege on the package. A routine library or executable can be comprised of multiple packages and therefore can have multiple package owners associated with it.

Routine Definer

The ID that issues the CREATE statement to register a routine. The routine definer is generally a DBA, but is also often the routine package owner. When a routine is invoked, at package load time, the authorization to run the routine is checked against the definer's authorization to execute the package or packages associated with the routine (not against the authorization of the routine invoker). For a routine to be successfully invoked, the routine definer must have one of:

- EXECUTE privilege on the package or packages of the routine and EXECUTE privilege on the routine
- DATAACCESS authority

If the routine definer and the routine package owner are the same user, then the routine definer will have the required EXECUTE privileges on the packages. If the definer is not the package owner, the definer must be explicitly granted EXECUTE privilege on the packages by any user with ACCESSCTRL or SECADM authority, CONTROL or EXECUTE WITH GRANT OPTION privilege on the package. (The creator of a package automatically receives CONTROL and EXECUTE WITH GRANT OPTION on the package.)

Upon issuing the CREATE statement that registers the routine, the definer is implicitly granted the EXECUTE WITH GRANT OPTION privilege on the routine.

The routine definer's role is to encapsulate under one authorization ID, the privileges of running the packages associated with a routine and the privilege of granting EXECUTE privilege on the routine to PUBLIC or to specific users that need to invoke the routine.

Note: For SQL routines the routine definer is also implicitly the package owner. Therefore the definer will have EXECUTE WITH GRANT OPTION on both the routine and on the routine package upon execution of the CREATE statement for the routine.

Routine Invoker

The ID that invokes the routine. To determine which users will be invokers of a routine, it is necessary to consider how a routine can be invoked. Routines can be invoked from a command window or from within an embedded SQL application. In the case of methods and UDFs the routine reference will be embedded in another SQL statement. A procedure is

invoked by using the CALL statement. For dynamic SQL in an application, the invoker is the runtime authorization ID of the immediately higher-level routine or application containing the routine invocation (however, this ID can also depend on the DYNAMICRULES option with which the higher-level routine or application was bound). For static SQL, the invoker is the value of the OWNER precompile/BIND option of the package that contains the reference to the routine. To successfully invoke the routine, these users will require EXECUTE privilege on the routine. This privilege can be granted by any user with EXECUTE WITH GRANT OPTION privilege on the routine (this includes the routine definer unless the privilege has been explicitly revoked), ACCESSCTRL, or SECADM authority, by explicitly issuing a GRANT statement.

As an example, if a package associated with an application containing dynamic SQL was bound with DYNAMICRULES BIND, then its runtime authorization ID will be its package owner, not the person invoking the package. Also, the package owner will be the actual binder or the value of the OWNER precompile/bind option. In this case, the invoker of the routine assumes this value rather than the ID of the user who is executing the application.

Note:

1. For static SQL within a routine, the package owner's privileges must be sufficient to execute the SQL statements in the routine body. These SQL statements might require table access privileges or execute privileges if there are any nested references to routines.
2. For dynamic SQL within a routine, the userid whose privileges will be validated are governed by the DYNAMICRULES option of the BIND of the routine body.
3. The routine package owner must GRANT EXECUTE on the package to the routine definer. This can be done before or after the routine is registered, but it must be done before the routine is invoked otherwise an error (SQLSTATE 42051) will be returned.

The steps involved in managing the execute privilege on a routine are detailed in the diagram and text that follows:

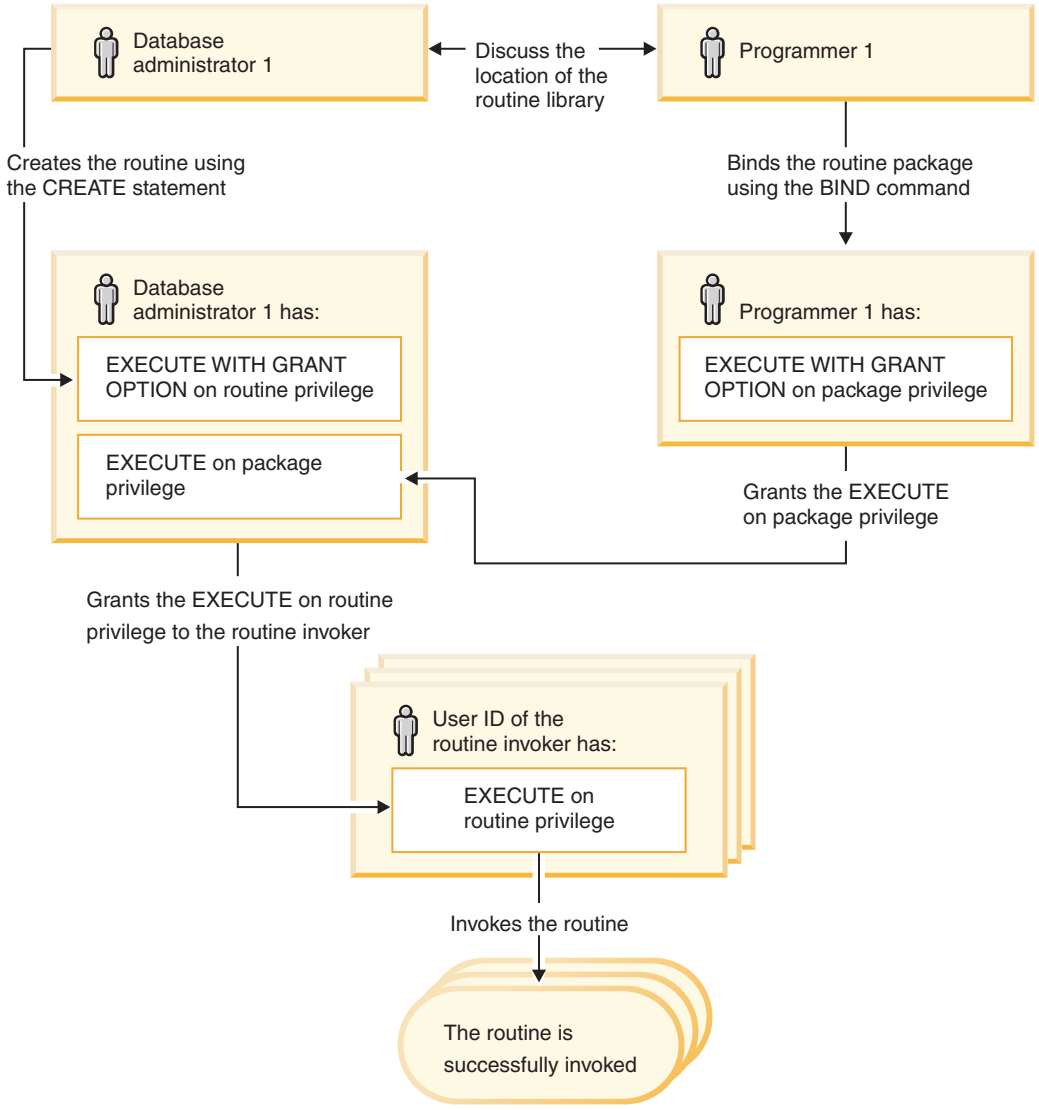


Figure 19. Managing the EXECUTE privilege on routines

1. Definer performs the appropriate CREATE statement to register the routine. This registers the routine in DB2 with its intended level of SQL access, establishes the routine signature, and also points to the routine executable. The definer, if not also the package owner, needs to communicate with the package owners and authors of the routine programs to be clear on where the routine libraries reside so that this can be correctly specified in the EXTERNAL clause of the CREATE statement. By virtue of a successful CREATE statement, the definer has EXECUTE WITH GRANT privilege on the routine, however the definer does not yet have EXECUTE privilege on the packages of the routine.
2. Definer must grant EXECUTE privilege on the routine to any users who are to be permitted use of the routine. (If the package for this routine will recursively call this routine, then this step must be done before the next step.)
3. Package owners precompile and bind the routine program, or have it done on their behalf. Upon a successful precompile and bind, the package owner is implicitly granted EXECUTE WITH GRANT OPTION privilege on the respective package. This step follows step one in this list only to cover the

possibility of SQL recursion in the routine. If such recursion does not exist in any particular case, the precompile/bind could precede the issuing of the CREATE statement for the routine.

4. Each package owner must explicitly grant EXECUTE privilege on their respective routine package to the definer of the routine. This step must come at some time after the previous step. If the package owner is also the routine definer, this step can be skipped.
5. Static usage of the routine: the bind owner of the package referencing the routine must have been given EXECUTE privilege on the routine, so the previous step must be completed at this point. When the routine executes, DB2 verifies that the definer has the EXECUTE privilege on any package that is needed, so step 3 must be completed for each such package.
6. Dynamic usage of the routine: the authorization ID as controlled by the DYNAMICRULES option for the invoking application must have EXECUTE privilege on the routine (step 4), and the definer of the routine must have the EXECUTE privilege on the packages (step 3).

Controlling database access

One of the most important responsibilities of the database administrator and the system administrator is database security.

Securing your database involves several activities:

- Preventing accidental loss of data or data integrity through equipment or system malfunction.
- Preventing unauthorized access to valuable data. You must ensure that sensitive information is not accessed by those without a “need to know”.
- Preventing unauthorized persons from committing mischief through malicious deletion or tampering with data.
- Monitoring access of data by users.

Planning for Security: Start by defining your objectives for a database access control plan, and specifying who shall have access to what and under what circumstances. Your plan should also describe how to meet these objectives by using database functions, functions of other programs, and administrative procedures.

Security considerations when installing and using the DB2 database manager

Security considerations are important to the DB2 administrator from the moment the product is installed.

To complete the installation of the DB2 database manager, a user ID, a group name, and a password are required. The GUI-based DB2 database manager install program creates default values for different user IDs and the group. Different defaults are created, depending on whether you are installing on Linux and UNIX or Windows platforms:

- On UNIX and Linux platforms, if you choose to create a DB2 instance in the instance setup window, the DB2 database install program creates, by default, different users for the DAS (`dasusr`), the instance owner (`db2inst`), and the fenced user (`db2fenc`). Optionally, you can specify different user names

The DB2 database install program appends a number from 1-99 to the default user name, until a user ID that does not already exist can be created. For example, if the users `db2inst1` and `db2inst2` already exist, the DB2 database install program creates the user `db2inst3`. If a number greater than 10 is used, the character portion of the name is truncated in the default user ID. For example, if the user ID `db2fenc9` already exists, the DB2 database install program truncates the `c` in the user ID, then appends the `10` (`db2fen10`). Truncation does not occur when the numeric value is appended to the default DAS user (for example, `dasusr24`).

- On Windows platforms, the DB2 database install program creates, by default, the user `db2admin` for the DAS user, the instance owner, and fenced users (you can specify a different user name during setup, if you want). Unlike Linux and UNIX platforms, no numeric value is appended to the user ID.

To minimize the risk of a user other than the administrator from learning of the defaults and using them in an improper fashion within databases and instances, change the defaults during the install to a new or existing user ID of your choice.

Note: Response file installations do not use default values for user IDs or group names. These values must be specified in the response file.

Passwords are very important when authenticating users. If no authentication requirements are set at the operating system level and the database is using the operating system to authenticate users, users will be allowed to connect. For example on Linux and UNIX operating systems, undefined passwords are treated as NULL. In this situation, any user without a defined password will be considered to have a NULL password. From the operating system's perspective, this is a match and the user is validated and able to connect to the database. Use passwords at the operating system level if you want the operating system to do the authentication of users for your database.

When working with DB2 Database Partitioning Feature (DPF) on Linux and UNIX operating system environments, the DB2 database manager by default uses the rsh utility (remsh on HP-UX) to run some commands on remote nodes. The rsh utility transmits passwords in clear text over the network, which can be a security exposure if the DB2 server is not on a secure network. You can use the DB2RSHCMD registry variable to set the remote shell program to a more secure alternative that avoids this exposure. One example of a more secure alternative is ssh. See the DB2RSHCMD registry variable documentation for restrictions on remote shell configurations.

After installing the DB2 database manager, also review, and change (if required), the default privileges that have been granted to users. By default, the installation process grants system administration (SYSADM) privileges to the following users on each operating system:

Linux and UNIX platforms
: To a valid DB2 database user name that belongs to the primary group of the instance owner.

Windows environments

- To members of the local Administrators group.
- If the DB2 database manager is configured to enumerate groups for users at the location where the users are defined, to members of the Administrators group at the Domain Controller. You use the

DB2_GRP_LOOKUP environment variable to configure group enumeration on Windows platforms.

- If Windows extended security is enabled, to members of the DB2ADMNS group. The location of the DB2ADMNS group is decided during installation.
- To the LocalSystem account

By updating the database manager configuration parameter **sysadm_group**, the administrator can control which group of users possesses SYSADM privileges. You must follow the guidelines below to complete the security requirements for both the DB2 database installation and the subsequent instance and database creation.

Any group defined as the system administration group (by updating **sysadm_group**) must exist. The name of this group should allow for easy identification as the group created for instance owners. User IDs and groups that belong to this group have system administrator authority for their respective instances.

The administrator should consider creating an instance owner user ID that is easily recognized as being associated with a particular instance. This user ID should have as one of its groups the name of the SYSADM group created above. Another recommendation is to use this instance-owner user ID only as a member of the instance owner group and not to use it in any other group. This should control the proliferation of user IDs and groups that can modify the instance.

The created user ID must be associated with a password to provide authentication before being permitted entry into the data and databases within the instance. The recommendation when creating a password is to follow your organization's password naming guidelines.

Note: To avoid accidentally deleting or overwriting instance configuration or other files, administrators should consider using another user account, which does not belong to the same primary group as the instance owner, for day-to-day administration tasks that are performed on the server directly.

Authentication methods for your server

Access to an instance or a database first requires that the user be *authenticated*. The *authentication type* for each instance determines how and where a user will be verified.

The authentication type is stored in the configuration file at the server. It is initially set when the instance is created. There is one authentication type per instance, which covers access to that database server and all the databases under its control.

If you intend to access data sources from a federated database, you must consider data source authentication processing and definitions for federated authentication types.

Note: You can check the following web site for certification information on the cryptographic routines used by the DB2 database management system to perform encryption of the userid and password when using SERVER_ENCRYPT authentication, and of the userid, password and user data when using DATA_ENCRYPT authentication: http://www.ibm.com/security/standards/st_evaluations.shtml.

Switching User on an Explicit Trusted Connection

For CLI/ODBC and XA CLI/ODBC applications, the authentication mechanism used when processing a switch user request that requires authentication is the same as the mechanism used to originally establish the trusted connection itself. Therefore, any other negotiated security attributes (for example, encryption algorithm, encryption keys, and plug-in names) used during the establishment of the explicit trusted connection are assumed to be the same for any authentication required for a switch user request on that trusted connection. JAVA applications allow the authentication method to be changed on a switch user request (by use of a datasource property).

Because a trusted context object can be defined such that switching user on a trusted connection does *not* require authentication, in order to take full advantage of the switch user on an explicit trusted connection feature, user-written security plug-ins must be able to:

- Accept a user ID-only token
- Return a valid DB2 authorization ID for that user ID

Note: An explicit trusted connection cannot be established if the CLIENT type of authentication is in effect.

Authentication types provided

The following authentication types are provided:

SERVER
Specifies that authentication occurs on the server through the security mechanism in effect for that configuration, for example, through a security plug-in module. The default security mechanism is that if a user ID and password are specified during the connection or attachment attempt, they are sent to the server and compared to the valid user ID and password combinations at the server to determine if the user is permitted to access the instance.

Note: The server code detects whether a connection is local or remote. For local connections, when authentication is SERVER, a user ID and password are not required for authentication to be successful.

SERVER_ENCRYPT
Specifies that the server accepts encrypted SERVER authentication schemes. If the client authentication is not specified, the client is authenticated using the method selected at the server. The user ID and password are encrypted when they are sent over the network from the client to the server.

When the resulting authentication method negotiated between the client and server is SERVER_ENCRYPT, you can choose to encrypt the user ID and password using an AES (Advanced Encryption Standard) 256–bit algorithm. To do this, set the **alternate_auth_enc** database manager configuration parameter. This configuration parameter has three settings:

- NOT_SPECIFIED (default) means that the server accepts the encryption algorithm that the client proposes, including an AES 256-bit algorithm.
- AES_CMP means that if the connecting client proposes DES but supports AES encryption, the server renegotiates for AES encryption.
- AES_ONLY means that the server only accepts AES encryption. If the client does not support AES encryption, the connection is rejected.

AES encryption can be used only when the authentication method negotiated between the client and server is SERVER_ENCRYPT.

CLIENT

Specifies that authentication occurs on the database partition where the application is invoked using operating system security. The user ID and password specified during a connection or attachment attempt are compared with the valid user ID and password combinations on the client node to determine if the user ID is permitted access to the instance. No further authentication will take place on the database server. This is sometimes called single signon.

If the user performs a local or client login, the user is known only to that local client workstation.

If the remote instance has CLIENT authentication, two other parameters determine the final authentication type: *trust_allclnts* and *trust_clntauth*.

CLIENT level security for TRUSTED clients only:

Trusted clients are clients that have a reliable, local security system.

When the authentication type of CLIENT has been selected, an additional option may be selected to protect against clients whose operating environment has no inherent security.

To protect against unsecured clients, the administrator can select Trusted Client Authentication by setting the *trust_allclnts* parameter to NO. This implies that all trusted platforms can authenticate the user on behalf of the server. Untrusted clients are authenticated on the Server and must provide a user ID and password. You use the *trust_allclnts* configuration parameter to indicate whether you are trusting clients. The default for this parameter is YES.

Note: It is possible to trust all clients (*trust_allclnts* is YES) yet have some of those clients as those who do not have a native safe security system for authentication.

You may also want to complete authentication at the server even for trusted clients. To indicate where to validate trusted clients, you use the *trust_clntauth* configuration parameter. The default for this parameter is CLIENT.

Note: For trusted clients only, if no user ID or password is explicitly provided when attempting to CONNECT or ATTACH, then validation of the user takes place at the client. The *trust_clntauth* parameter is only used to determine where to validate the information provided on the USER or USING clauses.

To protect against all clients, including JCC type 4 clients on z/OS and System i® but excluding native DB2 clients on z/OS, OS/390®, VM, VSE and System i, set the *trust_allclnts* parameter to DRDAONLY. Only these clients can be trusted to perform client-side authentication. All other clients must provide a user ID and password to be authenticated by the server.

The *trust_clntauth* parameter is used to determine where the above clients are authenticated: if *trust_clntauth* is "client", authentication takes place at the client. If *trust_clntauth* is "server", authentication takes place at the client when no user ID and password are provided and at the server when a user ID and password are provided.

Table 9. Authentication Modes using TRUST_ALLCLNTS and TRUST_CLNTAUTH Parameter Combinations.

TRUST_ ALLCLNTS	TRUST_ CLNTAUTH	Untrusted non–DRDA® Client Authen-tication (no user ID & password)	Untrusted non–DRDA Client Authen-tication (with user ID & password)	Trusted non–DRDA Client Authen-tication (no user ID & password)	Trusted non–DRDA Client Authen-tication (with user ID & password)	DRDA Client Authen-tication (no user ID & password)	DRDA Client Authen-tication (with user ID & password)
YES	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT
YES	SERVER	CLIENT	SERVER	CLIENT	SERVER	CLIENT	SERVER
NO	CLIENT	SERVER	SERVER	CLIENT	CLIENT	CLIENT	CLIENT
NO	SERVER	SERVER	SERVER	CLIENT	SERVER	CLIENT	SERVER
DRDAONLY	CLIENT	SERVER	SERVER	SERVER	SERVER	CLIENT	CLIENT
DRDAONLY	SERVER	SERVER	SERVER	SERVER	SERVER	CLIENT	SERVER

DATA_ENCRYPT

The server accepts encrypted SERVER authentication schemes and the encryption of user data. The authentication works exactly the same way as that shown with SERVER_ENCRYPT. The user ID and password are encrypted when they are sent over the network from the client to the server.

The following user data are encrypted when using this authentication type:

- SQL and XQuery statements.
- SQL program variable data.
- Output data from the server processing of an SQL or XQuery statement and including a description of the data.
- Some or all of the answer set data resulting from a query.
- Large object (LOB) data streaming.
- SQLDA descriptors.

DATA_ENCRYPT_CMP

The server accepts encrypted SERVER authentication schemes and the encryption of user data. In addition, this authentication type allows compatibility with down level products not supporting DATA_ENCRYPT authentication type. These products are permitted to connect with the SERVER_ENCRYPT authentication type and without encrypting user data. Products supporting the new authentication type must use it. This authentication type is only valid in the server's database manager configuration file and is not valid when used on the CATALOG DATABASE command.

KERBEROS

Used when both the DB2 client and server are on operating systems that support the Kerberos security protocol. The Kerberos security protocol performs authentication as a third party authentication service by using conventional cryptography to create a shared secret key. This key becomes a user's credential and is used to verify the identity of users during all occasions when local or network services are requested. The key eliminates the need to pass the user name and password across the network as clear text. Using the Kerberos security protocol enables the use of a single sign-on to a remote DB2 database server. The KERBEROS authentication

type is supported on various operating systems, refer to the related information section for more information.

Kerberos authentication works as follows:

1. A user logging on to the client machine using a domain account authenticates to the Kerberos key distribution center (KDC) at the domain controller. The key distribution center issues a ticket-granting ticket (TGT) to the client.
2. During the first phase of the connection the server sends the target principal name, which is the service account name for the DB2 database server service, to the client. Using the server's target principal name and the target-granting ticket, the client requests a service ticket from the ticket-granting service (TGS) which also resides at the domain controller. If both the client's ticket-granting ticket and the server's target principal name are valid, the TGS issues a service ticket to the client. The principal name recorded in the database directory may be specified as name/instance@REALM. (This is in addition to DOMAIN\userID and userID@xxx.xxx.xxx.com formats accepted on Windows.)
3. The client sends this service ticket to the server using the communication channel (which may be, as an example, TCP/IP).
4. The server validates the client's server ticket. If the client's service ticket is valid, then the authentication is completed.

It is possible to catalog the databases on the client machine and explicitly specify the Kerberos authentication type with the server's target principal name. In this way, the first phase of the connection can be bypassed.

If a user ID and a password are specified, the client will request the ticket-granting ticket for that user account and use it for authentication.

KRB_SERVER_ENCRYPT
Specifies that the server accepts KERBEROS authentication or encrypted SERVER authentication schemes. If the client authentication is KERBEROS, the client is authenticated using the Kerberos security system. If the client authentication is SERVER_ENCRYPT, the client is authenticated using a user ID and encryption password. If the client authentication is not specified, then the client will use Kerberos if available, otherwise it will use password encryption. For other client authentication types, an authentication error is returned. The authentication type of the client cannot be specified as KRB_SERVER_ENCRYPT

Note: The Kerberos authentication types are supported on clients and servers running on specific operating systems, refer to the related information section for more information. For Windows operating systems, both client and server machines must either belong to the same Windows domain or belong to trusted domains. This authentication type should be used when the server supports Kerberos and some, but not all, of the client machines support Kerberos authentication.

GSSPLUGIN
Specifies that the server uses a GSS-API plug-in to perform authentication. If the client authentication is not specified, the server returns a list of server-supported plug-ins, including any Kerberos plug-in that is listed in the *srvcon_gssplugin_list* database manager configuration parameter, to the client. The client selects the first plug-in found in the client plug-in directory from the list. If the client does not support any plug-in in the list,

the client is authenticated using the Kerberos authentication scheme (if it is returned). If the client authentication is the GSSPLUGIN authentication scheme, the client is authenticated using the first supported plug-in in the list.

GSS_SERVER_ENCRYPT
Specifies that the server accepts plug-in authentication or encrypted server authentication schemes. If client authentication occurs through a plug-in, the client is authenticated using the first client-supported plug-in in the list of server-supported plug-ins.

If the client authentication is not specified and an implicit connect is being performed (that is, the client does not supply a user ID and password when making the connection), the server returns a list of server-supported plug-ins, the Kerberos authentication scheme (if one of the plug-ins in the list is Kerberos-based), and the encrypted server authentication scheme. The client is authenticated using the first supported plug-in found in the client plug-in directory. If the client does not support any of the plug-ins that are in the list, the client is authenticated using the Kerberos authentication scheme. If the client does not support the Kerberos authentication scheme, the client is authenticated using the encrypted server authentication scheme, and the connection will fail because of a missing password. A client supports the Kerberos authentication scheme if a DB2-supplied Kerberos plug-in exists for the operating system, or a Kerberos-based plug-in is specified for the *srvcon_gssplugin_list* database manager configuration parameter.

If the client authentication is not specified and an explicit connection is being performed (that is, both the user ID and password are supplied), the authentication type is equivalent to SERVER_ENCRYPT. In this case, the choice of the encryption algorithm used to encrypt the user ID and password depends on the setting of the **alternate_auth_enc** database manager configuration parameter.

Note:

1. Do not inadvertently lock yourself out of your instance when you are changing the authentication information, since access to the configuration file itself is protected by information in the configuration file. The following database manager configuration file parameters control access to the instance:
   - AUTHENTICATION *
   - SYSADM_GROUP *
   - TRUST_ALLCLNTS
   - TRUST_CLNTAUTH
   - SYSCTRL_GROUP
   - SYSMAINT_GROUP

   * Indicates the two most important parameters.

   There are some things that can be done to ensure this does not happen: If you do accidentally lock yourself out of the DB2 database system, you have a fail-safe option available on all platforms that will allow you to override the usual DB2 database security checks to update the database manager configuration file using a highly privileged local operating system security user. This user *always* has the privilege to update the database manager configuration file and thereby correct the problem. However, this security bypass is restricted to a local update of the database manager configuration file.

You cannot use a fail-safe user remotely or for any other DB2 database command. This special user is identified as follows:

- UNIX platforms: the instance owner
- Windows platform: someone belonging to the local "Administrators" group
- Other platforms: there is no local security on the other platforms, so all users pass local security checks anyway

Authentication considerations for remote clients

When you catalog a database for remote access, you can specify the authentication type in the database directory entry.

The authentication type is not required and if not specified, the client defaults to SERVER_ENCRYPT. However, if the server does not support SERVER_ENCRYPT, the client attempts to retry using a value supported by the server. If the server supports multiple authentication types, the client will not choose among them, but instead returns an error. The error is returned to ensure that the correct authentication type is used. In this case, the client must catalog the database using a supported authentication type. If an authentication type is specified, authentication can begin immediately provided that value specified matches that at the server. If a mismatch is detected, DB2 database attempts to recover. Recovery may result in more flows to reconcile the difference, or in an error if the DB2 database cannot recover. In the case of a mismatch, the value at the server is assumed to be correct.

The authentication type DATA_ENCRYPT_CMP is designed to allow clients from a previous release that does not support data encryption to connect to a server using SERVER_ENCRYPT authentication instead of DATA_ENCRYPT. This authentication does not work when the following statements are true:

- The client level is Version 7.2.
- The gateway level is Version 8 FixPak7 or later.
- The server is Version 8 FixPak 7 or later.

When these are all true, the client cannot connect to the server. To allow the connection, you must either upgrade your client to Version 8 or later, or have your gateway level at Version 8 FixPak 6 or earlier.

The determination of the authentication type used when connecting is made by specifying the appropriate authentication type as a database catalog entry at the gateway. This is true for both DB2® Connect™ scenarios and for clients and servers in a partitioned database environment where the client has set the DB2NODE registry variable. You will catalog the authentication type at the catalog partition with the intent to "hop" to the appropriate partition. In this scenario, the authentication type cataloged at the gateway is not used because the negotiation is solely between the client and the server.

You may have a need to catalog multiple database aliases at the gateway using different authentication types if they need to have clients that use differing authentication types. When deciding which authentication type to catalog at a gateway, you can keep the authentication type the same as that used at the client and server; or, you can use the NOTSPEC authentication type with the understanding that NOTSPEC defaults to SERVER.

Details on controlling access to database objects

The control of access to database objects is through the use of GRANT and REVOKE statements. Implicit access authorization and indirect privileges are also discussed.

Granting privileges

To grant privileges on most database objects, you must have ACCESSCTRL authority, SECADM authority, or CONTROL privilege on that object; or, you must hold the privilege WITH GRANT OPTION. Additionally, users with SYSADM or SYSCTRL authority can grant table space privileges. You can grant privileges only on existing objects.

About this task

To grant CONTROL privilege to someone else, you must have ACCESSCTRL or SECADM authority. To grant ACCESSCTRL, DATAACCESS, DBADM or SECADM authority, you must have SECADM authority.

The GRANT statement allows an authorized user to grant privileges. A privilege can be granted to one or more authorization names in one statement; or to PUBLIC, which makes the privileges available to all users. Note that an authorization name can be either an individual user or a group.

On operating systems where users and groups exist with the same name, you should specify whether you are granting the privilege to the user or group. Both the GRANT and REVOKE statements support the keywords USER, GROUP, and ROLE. If these optional keywords are not used, the database manager checks the operating system security facility to determine whether the authorization name identifies a user or a group; it also checks whether an authorization ID of type role with the same name exists. If the database manager cannot determine whether the authorization name refers to a user, a group, or a role, an error is returned. The following example grants SELECT privileges on the EMPLOYEE table to the user HERON:

```
GRANT SELECT
 ON EMPLOYEE TO USER HERON
```

The following example grants SELECT privileges on the EMPLOYEE table to the group HERON:

```
GRANT SELECT
   ON EMPLOYEE TO GROUP HERON
```

In the Control Center, you can use the Schema Privileges notebook, the Table Space Privileges notebook, and the View Privileges notebook to grant and revoke privileges for these database objects. To open one of these notebooks, follow these steps:

1. In the Control Center, expand the object tree until you find the folder containing the objects you want to work with, for example, the **Views** folder.
2. Click the folder.

   Any existing database objects in this folder are displayed in the contents pane.
3. Right-click the object of interest in the contents pane and select **Privileges** in the pop-up menu.

   The appropriate Privileges notebook opens.

Revoking privileges

The REVOKE statement allows authorized users to revoke privileges previously granted to other users.

About this task

To revoke privileges on database objects, you must have ACCESSCTRL authority, SECADM authority, or CONTROL privilege on that object. Table space privileges can also be revoked by users with SYSADM and SYSCTRL authority. Note that holding a privilege WITH GRANT OPTION is not sufficient to revoke that privilege. To revoke CONTROL privilege from another user, you must have ACCESSCTRL, or SECADM authority. To revoke ACCESSCTRL, DATAACCESS, DBADM or SECADM authority, you must have SECADM authority. Table space privileges can be revoked only by a user who holds SYSADM, or SYSCTRL authority. Privileges can only be revoked on existing objects.

Note: A user without ACCESSCTRL authority, SECADM authority, or CONTROL privilege is not able to revoke a privilege that they granted through their use of the WITH GRANT OPTION. Also, there is no cascade on the revoke to those who have received privileges granted by the person being revoked.
If an explicitly granted table (or view) privilege is revoked from a user with DBADM authority, privileges will not be revoked from other views defined on that table. This is because the view privileges are available through the DBADM authority and are not dependent on explicit privileges on the underlying tables.

If a privilege has been granted to a user, a group, or a role with the same name, you must specify the GROUP, USER, or ROLE keyword when revoking the privilege. The following example revokes the SELECT privilege on the EMPLOYEE table from the user HERON:

```
REVOKE SELECT
   ON EMPLOYEE FROM USER HERON
```

The following example revokes the SELECT privilege on the EMPLOYEE table from the group HERON:

```
REVOKE SELECT
   ON EMPLOYEE FROM GROUP HERON
```

Note that revoking a privilege from a group may not revoke it from all members of that group. If an individual name has been directly granted a privilege, it will keep it until that privilege is directly revoked.

If a table privilege is revoked from a user, privileges are also revoked on any view created by that user which depends on the revoked table privilege. However, only the privileges implicitly granted by the system are revoked. If a privilege on the view was granted directly by another user, the privilege is still held.

If a table privilege is revoked from a user, privileges are also revoked on any view created by that user which depends on the revoked table privilege. However, only the privileges implicitly granted by the system are revoked. If a privilege on the view was granted directly by another user, the privilege is still held.

You may have a situation where you want to GRANT a privilege to a group and then REVOKE the privilege from just one member of the group. There are only a couple of ways to do that without receiving the error message SQL0556N:

- You can remove the member from the group; or, create a new group with fewer members and GRANT the privilege to the new group.
- You can REVOKE the privilege from the group and then GRANT it to individual users (authorization IDs).

Note: When CONTROL privilege is revoked from a user on a table or a view, the user continues to have the ability to grant privileges to others. When given CONTROL privilege, the user also receives all other privileges WITH GRANT OPTION. Once CONTROL is revoked, all of the other privileges remain WITH GRANT OPTION until they are explicitly revoked.

All packages that are dependent on revoked privileges are marked invalid, but can be validated if rebound by a user with appropriate authority. Packages can also be rebuilt if the privileges are subsequently granted again to the binder of the application; running the application will trigger a successful implicit rebind. If privileges are revoked from PUBLIC, all packages bound by users having only been able to bind based on PUBLIC privileges are invalidated. If DBADM authority is revoked from a user, all packages bound by that user are invalidated including those associated with database utilities. Attempting to use a package that has been marked invalid causes the system to attempt to rebind the package. If this rebind attempt fails, an error occurs (SQLCODE -727). In this case, the packages must be explicitly rebound by a user with:

- Authority to rebind the packages
- Appropriate authority for the objects used within the packages

These packages should be rebound at the time the privileges are revoked.

If you define a trigger or SQL function based on one or more privileges and you lose one or more of these privileges, the trigger or SQL function cannot be used.

Managing implicit authorizations by creating and dropping objects

The database manager implicitly grants certain privileges to a user that creates a database object such as a table or a package. Privileges are also granted when objects are created by users with DBADM authority. Similarly, privileges are removed when an object is dropped.

About this task

When the created object is a table, nickname, index, or package, the user receives CONTROL privilege on the object. When the object is a view, the CONTROL privilege for the view is granted implicitly only if the user has CONTROL privilege for all tables, views, and nicknames referenced in the view definition.

When the object explicitly created is a schema, the schema owner is given ALTERIN, CREATEIN, and DROPIN privileges WITH GRANT OPTION. An implicitly created schema has CREATEIN granted to PUBLIC.

Establishing ownership of a package

The BIND and PRECOMPILE commands create or change an application package. On either one, use the **OWNER** option to name the owner of the resulting package.

About this task

There are simple rules for naming the owner of a package:

- Any user can name themselves as the owner. This is the default if the **OWNER** option is not specified.
- A user ID with DBADM authority can name any authorization ID as the owner using the **OWNER** option.

Not all operating systems that can bind a package using DB2 database products support the **OWNER** option.

Implicit privileges through a package

Access to data within a database can be requested by application programs, as well as by persons engaged in an interactive workstation session. A package contains statements that allow users to perform a variety of actions on many database objects. Each of these actions requires one or more privileges.

Privileges granted to individuals binding the package and to PUBLIC, as well as to the roles granted to the individuals and to PUBLIC, are used for authorization checking when static SQL and XQuery statements are bound. Privileges granted through groups, and the roles granted to groups, are not used for authorization checking when static SQL and XQuery statements are bound.

Unless VALIDATE RUN is specified when binding the package, the user with a valid authorization ID who binds a package must either:

- Have been granted all the privileges required to execute the static SQL or XQuery statements in the package.
- Have acquired the necessary privileges through membership in one or more of:
  - PUBLIC
  - The roles granted to PUBLIC
  - The roles granted to the user

If VALIDATE RUN is specified at BIND time, all authorization failures for any static SQL or XQuery statements within this package will not cause the BIND to fail, and those SQL or XQuery statements are revalidated at run time. PUBLIC, group, role, and user privileges are all used when checking to ensure the user has the appropriate authorization (BIND or BINDADD privilege) to bind the package.

Packages may include both static and dynamic SQL and XQuery statements. To process a package with static queries, a user need only have EXECUTE privilege on the package. This user can then implicitly obtain the privileges of the package binder for any static queries in the package but only within the restrictions imposed by the package.

If the package includes dynamic SQL or XQuery statements, the required privileges depend on the value that was specified for **DYNAMICRULES** when the package was precompiled or bound. For more information, see the topic that describes the effect of **DYNAMICRULES** on dynamic queries.

Indirect privileges through a package containing nicknames

When a package contains references to nicknames, authorization processing for package creators and package users is slightly more complex.

When a package creator successfully binds packages that contain nicknames, the package creator does not have to pass authentication checking or privilege checking for the tables and views that the nicknames reference at the data source. However, the package executor must pass authentication and authorization checking at data sources.

For example, assume that a package creator's .SQC file contains several SQL or XQuery statements. One static statement references a local table. Another dynamic statement references a nickname. When the package is bound, the package creator's authid is used to verify privileges for the local table and the nickname, but no checking is done for the data source objects that the nickname identifies. When another user executes the package, assuming they have the EXECUTE privilege for that package, that user does not have to pass any additional privilege checking for the statement referencing the table. However, for the statement referencing the nickname, the user executing the package must pass authentication checking and privilege checking at the data source.

When the .SQC file contains only dynamic SQL and XQuery statements and a mixture of table and nickname references, DB2 database authorization checking for local objects and nicknames is similar. Package users must pass privilege checking for any local objects (tables, views) within the statements and also pass privilege checking for nickname objects (package users must pass authentication and privilege checking at the data source containing the objects that the nicknames identify). In both cases, users of the package must have the EXECUTE privilege.

The authorization ID and password of the package executor is used for all data source authentication and privilege processing. This information can be changed by creating a user mapping.

Note: Nicknames cannot be specified in static SQL and XQuery statements. Do not use the **DYNAMICRULES** option (set to BIND) with packages containing nicknames.

It is possible that packages containing nicknames might require additional authorization steps because DB2 database uses dynamic SQL when communicating with DB2 Family data sources. The authorization ID running the package at the data source must have the appropriate authority to execute the package dynamically at that data source.

Controlling access to data with views

A view provides a means of controlling access or extending privileges to a table.

Using a view allows the following kinds of control over access to a table:

- Access only to designated columns of the table.

  For users and application programs that require access only to specific columns of a table, an authorized user can create a view to limit the columns addressed only to those required.
- Access only to a subset of the rows of the table.

  By specifying a WHERE clause in the subquery of a view definition, an authorized user can limit the rows addressed through a view.
- Access only to a subset of the rows or columns in data source tables or views. If you are accessing data sources through nicknames, you can create local DB2 database views that reference nicknames. These views can reference nicknames from one or many data sources.

Note: Because you can create a view that contains nickname references for more than one data source, your users can access data in multiple data sources from one view. These views are called *multi-location views*. Such views are useful when joining information in columns of sensitive tables across a distributed environment or when individual users lack the privileges needed at data sources for specific objects.

To create a view, a user must have DATAACCESS authority, or CONTROL or SELECT privilege for each table, view, or nickname referenced in the view definition. The user must also be able to create an object in the schema specified for the view. That is, DBADM authority, CREATEIN privilege for an existing schema, or IMPLICIT_SCHEMA authority on the database if the schema does not already exist.

If you are creating views that reference nicknames, you do not need additional authority on the data source objects (tables and views) referenced by nicknames in the view; however, users of the view must have SELECT authority or the equivalent authorization level for the underlying data source objects when they access the view.

If your users do not have the proper authority at the data source for underlying objects (tables and views), you can:

1. Create a data source view over those columns in the data source table that are OK for the user to access
2. Grant the SELECT privilege on this view to users
3. Create a nickname to reference the view

Users can then access the columns by issuing a SELECT statement that references the new nickname.

The following scenario provides a more detailed example of how views can be used to restrict access to information.

Many people might require access to information in the STAFF table, for different reasons. For example:

- The personnel department needs to be able to update and look at the entire table.

  This requirement can be easily met by granting SELECT and UPDATE privileges on the STAFF table to the group PERSONNL:

  ```
  GRANT SELECT,UPDATE ON TABLE STAFF TO GROUP PERSONNL
  ```

- Individual department managers need to look at the salary information for their employees.

  This requirement can be met by creating a view for each department manager. For example, the following view can be created for the manager of department number 51:

  ```
  CREATE VIEW EMP051 AS
     SELECT NAME,SALARY,JOB FROM STAFF
     WHERE DEPT=51
  GRANT SELECT ON TABLE EMP051 TO JANE
  ```

  The manager with the authorization name JANE would query the EMP051 view just like the STAFF table. When accessing the EMP051 view of the STAFF table, this manager views the following information:

NAME	SALARY	JOB
Fraye	45150.0	Mgr
Williams	37156.5	Sales
Smith	35654.5	Sales
Lundquist	26369.8	Clerk
Wheeler	22460.0	Clerk

- All users need to be able to locate other employees. This requirement can be met by creating a view on the NAME column of the STAFF table and the LOCATION column of the ORG table, and by joining the two tables on their respective DEPT and DEPTNUMB columns:

```
CREATE VIEW EMPLOCS AS
   SELECT NAME, LOCATION FROM STAFF, ORG
   WHERE STAFF.DEPT=ORG.DEPTNUMB
GRANT SELECT ON TABLE EMPLOCS TO PUBLIC
```

  Users who access the employee location view will see the following information:

NAME	LOCATION
Molinare	New York
Lu	New York
Daniels	New York
Jones	New York
Hanes	Boston
Rothman	Boston
Ngan	Boston
Kermisch	Boston
Sanders	Washington
Pernal	Washington
James	Washington
Sneider	Washington
Marenghi	Atlanta
O'Brien	Atlanta
Quigley	Atlanta
Naughton	Atlanta
Abrahams	Atlanta
Koonitz	Chicago
Plotz	Chicago
Yamaguchi	Chicago
Scoutten	Chicago
Fraye	Dallas
Williams	Dallas
Smith	Dallas
Lundquist	Dallas
Wheeler	Dallas
Lea	San Francisco

NAME	LOCATION
Wilson	San Francisco
Graham	San Francisco
Gonzales	San Francisco
Burke	San Francisco
Quill	Denver
Davis	Denver
Edwards	Denver
Gafney	Denver

Controlling access for database administrators (DBAs)

You may want to monitor, control, or prevent access to data by database administrators (users holding DBADM authority).

Monitoring access to data

You can use the DB2 audit facility to monitor access by database administrators. To do so, follow these steps:

1. Create an audit policy that monitors the events you want to capture for users who hold DBADM authority.
2. Associate this audit policy with the DBADM authority.

Controlling access to data

You can use trusted contexts in conjunction with a role to control access by database administrators. To do so, follow these steps:

1. Create a role and grant DBADM authority to that role.
2. Define a trusted context and make the role the default role for this trusted context.

   Do not grant membership in the role to any authorization ID explicitly. This way, the role is available only through this trusted context and a user acquires DBADM capability only when they are within the confines of the trusted context.
3. There are two ways you can control how users access the trusted context:
   - Implicit access: Create a unique trusted context for each user. When the user establishes a regular connection that matches the attributes of the trusted context, they are implicitly trusted and gain access to the role.
   - Explicit access: Create a trusted context using the WITH USE FOR clause to define all users who can access it. Create an application through which those users can make database requests. The application establishes an explicit trusted connection, and when a user issues a request, the application switches to that user ID and executes the request as that user on the database.

If you want to monitor the use of this trusted context, you can create an audit policy that captures the events you are interested in for users of this trusted context. Associate this audit policy with the trusted context.

Preventing access to data

To prevent access to data in tables, choose one of these options:

- To prevent access to data in all tables, revoke DATAACCESS from your DBADM user, role or group. Alternatively, you could grant DBADM to the user, role or group of interest without the DATAACCESS option
- To prevent access to data in one particular table, follow these steps:
  - Assign a security label to every column in the table.
  - Grant that security label to a role.
  - Grant that role to all users (or roles) that have a legitimate need to access the table.

  No user, regardless of their authority, will be able to access data in that table unless they are a member in that role.

Data encryption

The DB2 database system offers several ways to encrypt data, both while in storage, and while in transit over the network.

Encrypting data in storage

You have the following options for encrypting data in storage:

- You can use the encryption and decryption built-in functions ENCRYPT, DECRYPT_BIN, DECRYPT_CHAR, and GETHINT to encrypt your data within database tables.
- You can use IBM Database Encryption Expert to encrypt the underlying operating system data and backup files.
- If you are running a DB2 Enterprise Server Edition system on the AIX operating system, and you are interested in file-level encryption only, you can use encrypted file system (EFS) to encrypt your operating system data and backup files.

Encrypting data in transit

To encrypt data in-transit between clients and DB2 databases, you can use the DATA_ENCRYPT authentication type, or, the DB2 database system support of Secure Sockets Layer (SSL).

Using the ENCRYPT, DECRYPT_BIN, DECRYPT_CHAR, and GETHINT functions

The ENCRYPT built-in function encrypts data using a password-based encryption method. These functions also allow you to encapsulate a password hint. The password hint is embedded in the encrypted data. Once encrypted, the only way to decrypt the data is by using the correct password. Developers that choose to use these functions should plan for the management of forgotten passwords and unusable data.

The result of the ENCRYPT functions is VARCHAR FOR BIT DATA (with a limit of 32631).

Only CHAR, VARCHAR, and FOR BIT DATA can be encrypted.

The DECRYPT_BIN and DECRYPT_CHAR functions decrypt data using password-based decryption.

DECRYPT_BIN always returns VARCHAR FOR BIT DATA while DECRYPT_CHAR always returns VARCHAR. Since the first argument may be CHAR FOR BIT DATA or VARCHAR FOR BIT DATA, there are cases where the result is not the same as the first argument.

The length of the result depends on the bytes to the next 8 byte boundary. The length of the result could be the length of the data argument plus 40 plus the number of bytes to the next 8 byte boundary when the optional hint parameter is specified. Or, the length of the result could be the length of the data argument plus 8 plus the number of bytes to the next 8 byte boundary when the optional hint parameter is not specified.

The GETHINT function returns an encapsulated password hint. A password hint is a phrase that will help data owners remember passwords. For example, the word “Ocean” can be used as a hint to remember the password ″Pacific″.

The password that is used to encrypt the data is determined in one of two ways:

- Password Argument. The password is a string that is explicitly passed when the ENCRYPT function is invoked. The data is encrypted and decrypted with the given password.
- Encryption password special register. The SET ENCRYPTION PASSWORD statement encrypts the password value and sends the encrypted password to the database manager to store in a special register. ENCRYPT, DECRYPT_BIN and DECRYPT_CHAR functions invoked without a password parameter use the value in the ENCRYPTION PASSWORD special register. The ENCRYPTION PASSWORD special register is only stored in encrypted form.

  The initial or default value for the special register is an empty string.

Valid lengths for passwords are between 6 and 127 inclusive. Valid lengths for hints are between 0 and 32 inclusive.

IBM Database Encryption Expert for encryption of data at rest

IBM Database Encryption Expert is a comprehensive software data security solution that when used in conjunction with native DB2 security provides effective protection of the data and the database application against a broad array of threats.

Database Encryption Expert helps organizations ensure that private and confidential data is strongly protected and in compliance with regulations and legislative acts. The key benefits of Database Encryption Expert are:

- Proven, strong data security for the DB2 database system
- Protection of live files, configuration files, log files and back-up data
- Transparent to application, database and storage environments
- Unified policy and key management for protecting data in both on-line and off-line environments
- Meets performance requirements

Database Encryption Expert enables you to encrypt offline database backups and to encrypt online (″live″) database files. This is encryption of data on the disk, sometimes called “data at rest” as opposed to ″data in flight″, which is travelling over the network.

- For backups, data is encrypted as it is being backed up, so the data on the backup device is encrypted. Should the data need to be recovered, the recovery server recognizes that the data is encrypted and will un-encrypt the data.
- For database files, the operating system data files containing the data from the DB2 database are encrypted. This protects the data files from unauthorized users trying to read the "raw" database file.

Database Encryption Expert is transparent to users, databases, applications, and storage. No code changes or changes to existing infrastructure are required. Database Encryption Expert can protect data in any storage environment, while users continue to access data the in the same way as before.

Database Encryption Expert can protect database applications, because it can prevent changes to executable files, configuration files, libraries, and so on, thereby preventing attacks on the application.

Architecture of Database Encryption Expert

Database Encryption Expert is a set of agent and server software packages that you administer by using a Web-based user-interface and command-line utilities. The Database Encryption Expert administrator configures security policies that govern how security and encryption are implemented.

According to how these security policies are defined, the Database Encryption Expert backup agent encrypts DB2 backups, and the Database Encryption Expert file system agent encrypts DB2 data files.

The Encryption Expert Security Server stores the security policies, encryption keys and event log files. Security policies contain sets of security rules that must be satisfied in order to allow or deny access. Each security rule evaluates who, what, when, and how protected data is accessed and, if these criteria match, the Security Server either permits or denies access.

Figure 20 on page 126 illustrates the architecture of Database Encryption Expert.

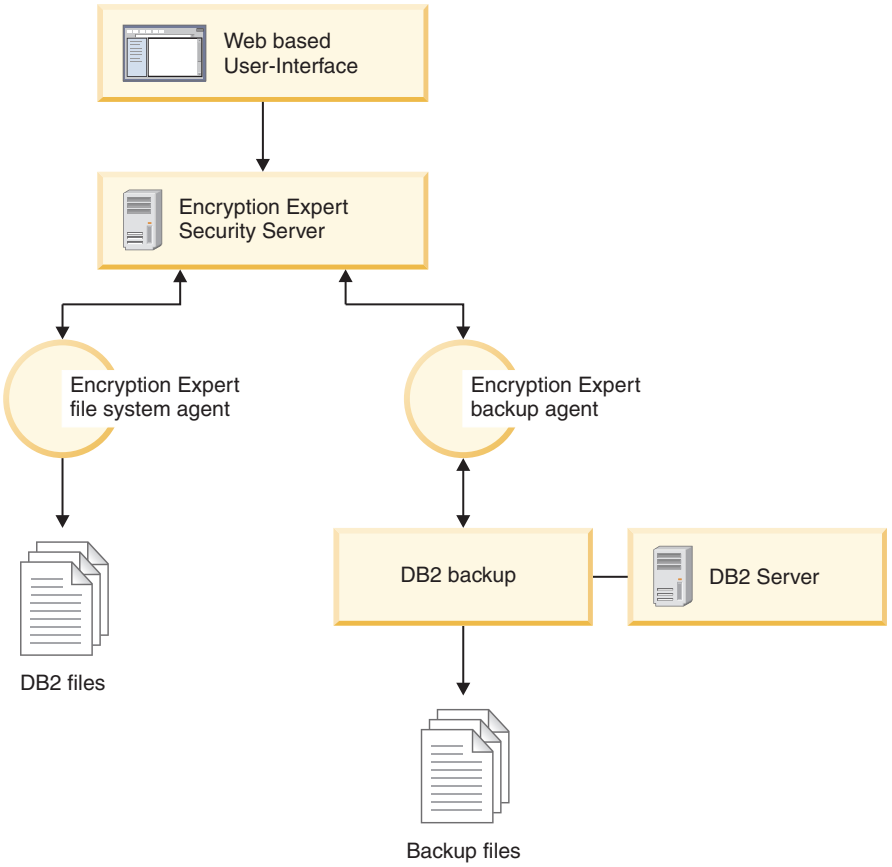


Figure 20. Architecture of Database Encryption Expert

File system agent

The Database Encryption Expert file system agent process is always running in the background. The agent intercepts any attempt to access data files, directories, or executables that you are protecting. The Database Encryption Expert file system agent forwards the access attempt to the Security Server and, based upon the applied policy, the Security Server grants or denies the attempted access.

Database Encryption Expert protection extends beyond simply allowing or denying access to a file, you can also encrypt files. Just the file contents is encrypted, but the file metadata is left intact. Therefore, you do not have to decrypt an encrypted file just to see it's name, timestamps, file type, and so on. This allows data management applications to perform their functions without exposing the file contents. For example, backup managers can backup specific data, without being able to see the contents.

If an encrypted file is accessed by an unauthorized user, its contents are worthless without the appropriate Security Server approval and encryption keys. However, users with the correct policies and permissions are unaware that encryption and decryption are taking place.

Backup agent

All database backup functions that are normally performed by the DB2 backup API system are supported by the Database Encryption Expert server, including native database compression. Other than an additional command-line argument,

DB2 backup operators are unaware of Database Encryption Expert intervention. Database Encryption Expert backs up and restores static data-at-rest and active online data.

Basic backup and restore configuration is supported. In the basic configuration, data is encrypted and backed up with one server and multiple agents; data is decrypted and restored on an agent that is configured with the same server that was originally used to make the backup.

Single-site and multi-site configurations are also supported for backup and restore. In a single-site scenario, configuration data is mirrored across multiple Security Servers in a single data center. In a multi-site scenario, backups are restored on different Encryption Expert servers in different data centers.

Audit logging

Database Encryption Expert agent activity is closely monitored and logged through a centralized audit logging facility. All auditable events, including backups, restores, and security administration operations can be logged. This includes Database Encryption Expert system events, such as initialization, shut down and restart; and network connects and disconnects between different Database Encryption Expert components.

Database Encryption Expert documentation

For more information about Database Encryption Expert, go to the following web page:http://publib.boulder.ibm.com/infocenter/mptoolic/v1r0/topic/com.ibm.db2tools.eet.doc.ug/eetwelcome.htm.

Secure Sockets Layer (SSL)

The DB2 database system supports the use of Secure Sockets Layer (SSL) and it's successor, Transport Layer Security (TLS), to enable a client to authenticate a server, and to provide private communication between the client and server by use of encryption. Authentication is performed by the exchange of digital certificates.

Note: When this topic mentions SSL, the same information applies to TLS, unless otherwise noted.

Without encryption, packets of information travel through networks in full view of anyone who has access. You can use SSL to protect data in transit on all networks that use TCP/IP (you can think of an SSL connection as a secured TCP/IP connection).

A client and server establish a secure SSL connection by performing an ″SSL handshake″.

Overview of the SSL handshake

During an *SSL handshake*, a public-key algorithm, usually RSA, is used to securely exchange digital signatures and encryption keys between a client and a server. This identity and key information is used to establish a secure connection for the session between the client and the server. After the secure session is established, data transmission between the client and server is encrypted using a symmetric algorithm, such as AES.

The client and server perform the following steps during the SSL handshake:

1. The client requests an SSL connection and lists its supported cipher suites.
2. The server responds with a selected cipher suite.
3. The server sends its digital certificate to the client.
4. The client verifies the validity of the server certificate, for authentication purposes. It can do this by checking with the trusted certificate authority that issued the server certificate or by checking in its own key database.
5. The client and server securely negotiate a session key and a message authentication code (MAC).
6. The client and server securely exchange information using the key and MAC selected.

Note: The DB2 database system does not support the (optional) authentication of the client during the SSL handshake.

Using SSL encryption with DB2 authentication

You can use SSL encryption in conjunction with all existing DB2 authentication methods, such as KERBEROS or SERVER. You do this as usual by setting the authentication type for the instance in the DBM configuration parameters to the authentication method of your choice.

Digital certificates and certificate authorities

Digital certificates are issued by trusted parties, called certificate authorities, to verify the identity of an entity, such as a client or server.

The digital certificate serves two purposes: it verifies the owner's identity and it makes the owner's public key available. It is issued with an expiration date, after which it is no longer guaranteed by the certificate authority (CA).

To obtain a digital certificate, you send a request to the CA of your choice, such as Verisign, or RSA. The request includes your distinguished name, your public key, and your signature. A distinguished name (DN) is a unique identifier for each user or host for which you are applying for a certificate. The CA checks your signature using your public key and performs some level of verification of your identity (this varies with different CAs). After verification, the CA sends you a signed digital certificate that contains your distinguished name, your public key, the CA's distinguished name, and the signature of the certificate authority. You store this signed certificate in your key database.

When you send this certificate to a receiver, the receiver performs two steps to verify your identity:

1. Uses your public key that comes with the certificate to check your digital signature.
2. Verifies that the CA that issued your certificate is legitimate and trustworthy. To do this, the receiver needs the public key of the CA. The receiver might already hold an assured copy of the public key of the CA in their key database, but if not, the receiver must acquire an additional digital certificate to obtain the public key of the CA. This certificate might in turn depend on the digital certificate of another CA; there might be a hierarchy of certificates issued by multiple CAs, each depending on the validity of the next. Eventually, however, the receiver needs the public key of the *root CA*. The root CA is the CA at the top of the hierarchy. To trust the validity of the digital certificate of the root

CA, the public-key user must receive that digital certificate in a secure manner, such as through a download from an authenticated server, or with preloaded software received from a reliable source, or on a securely delivered diskette.

Many applications that send a digital certificate to a receiver send not just their own certificate, but also all of the CA digital certificates necessary to verify the hierarchy of certificates up to the root CA certificate.

For a digital certificate to be entirely trustworthy, the owner of the digital certificate must have carefully protected their private key, for example, by encrypting it on their computer's hard drive. If their private key has been compromised, an imposter could misuse their digital certificate.

You can use self-signed digital certificates for testing purposes. A self-signed digital certificate contains your distinguished name, your public key, and your signature.

Public-key cryptography

SSL uses public-key algorithms to exchange encryption key information and digital certificate information for authentication. Public-key cryptography (also known as asymmetric cryptography) uses two different encryption keys: a public key to encrypt data and an associated private key to decrypt it.

Conversely, symmetric key cryptography uses just one key, which is shared by all parties involved in the secure communication. This secret key is used both to encrypt and decrypt information. The key must be safely distributed to, and stored by, all parties, which is difficult to guarantee. With public-key cryptography, the public key is not secret, but the messages it encrypts can only be decrypted by using it's associated private key. The private key must be securely stored, for example, in your key database, or encrypted on your computer's hard drive.

Public-key algorithms alone do not guarantee secure communication, you also need to verify the identity of whoever is communicating with you. To perform this authentication, SSL uses digital certificates. When you send your digital certificate to someone, the certificate provides them with your public key. You have used your private key to digitally sign your certificate and so the receiver of the communication can use your public key to verify your signature. The validity of the digital certificate itself is guaranteed by the certificate authority (CA) that issued it.

Supported cipher suites

During an SSL handshake, the client and server negotiate which cipher suite to use to exchange data. A cipher suite is a set of algorithms that are used to provide authentication, encryption, and data integrity.

The DB2 database system uses GSKit running in FIPS mode to provide SSL support. GSKit supports the following cipher suites:

- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA

The name of each cipher suite specifies the algorithms that it uses for authentication, encryption, and data integrity checking. For example, the cipher

suite TLS_RSA_WITH_AES_256_CBC_SHA uses RSA for authentication; AES 256-bit and CBC for encryption algorithms; and SHA-1 for the hash function for data integrity.

During the SSL handshake, the DB2 database system automatically picks the strongest cipher suite supported by both the client and the server. If you want the server to accept only one or more specific cipher suites, you can set the **ssl_cipherspecs** configuration parameter to any of the following values:

- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- Any combination of these three values. To set multiple values, separate each value by a comma but do not put a space between the values.
- Null. In this case, the strongest available algorithm is automatically picked.

You cannot prioritize which cipher suite is selected. If you set the **ssl_cipherspecs** configuration parameter, the DB2 database system picks the strongest available cipher suite; this selection does not depend on the order you specify the cipher suites when you set **ssl_cipherspecs**.

GSKit return codes

Some DB2 database manager messages might display a return code from the IBM Global Security Kit (GSKit).

General GSKit return codes

Table 10. GSKit general return codes

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x00000000	0	GSK_OK	The task completed successfully. Issued by every function call that completes successfully.
0x00000001	1	GSK_INVALID_HANDLE	The environment or SSL handle is not valid. The specified handle was not the result of a successful open function call.
0x00000002	2	GSK_API_NOT_AVAILABLE	The dynamic link library (DLL) has been unloaded and is not available. (Windows only.)
0x00000003	3	GSK_INTERNAL_ERROR	Internal error. Report this error to service.
0x00000004	4	GSK_INSUFFICIENT_STORAGE	Insufficient memory is available to perform the operation.
0x00000005	5	GSK_INVALID_STATE	The handle is in an invalid state for operation, such as performing an init operation on a handle twice.
0x00000006	6	GSK_KEY_LABEL_NOT_FOUND	Specified key label not found in key file.
0x00000007	7	GSK_CERTIFICATE_NOT_AVAILABLE	Certificate not received from partner.

Table 10. GSKit general return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x00000008	8	GSK_ERROR_CERT_VALIDATION	Certificate validation error.
0x00000009	9	GSK_ERROR_CRYPTO	Error processing cryptography.
0x0000000a	10	GSK_ERROR_ASN	Error validating ASN fields in certificate.
0x0000000b	11	GSK_ERROR_LDAP	Error connecting to LDAP server.
0x0000000c	12	GSK_ERROR_UNKNOWN_ERROR	Internal error. Report this error to service.
0x00000065	101	GSK_OPEN_CIPHER_ERROR	Internal error. Report this error to service.
0x00000066	102	GSK_KEYFILE_IO_ERROR	I/O error reading the key file.
0x00000067	103	GSK_KEYFILE_INVALID_FORMAT	The key file has an invalid internal format. Re-create the key file.
0x00000068	104	GSK_KEYFILE_DUPLICATE_KEY	The key file has two entries with the same key. Use the iKeyman utility to remove the duplicate key.
0x00000069	105	GSK_KEYFILE_DUPLICATE_LABEL	The key file has two entries with the same label. Use the iKeyman utility to remove the duplicate label.
0x0000006a	106	GSK_BAD_FORMAT_OR_ INVALID_PASSWORD	The key file password is used as an integrity check. Either the keyfile has become corrupted or the password ID is incorrect.
0x0000006b	107	GSK_KEYFILE_CERT_EXPIRED	The default key in the key file has an expired certificate. Use the iKeyman utility to remove certificates that are expired.
0x0000006c	108	GSK_ERROR_LOAD_GSKLIB	An error occurred loading one of the GSKit dynamic link libraries. Be sure GSKit was installed correctly.
0x0000006d	109	GSK_PENDING_CLOSE_ERROR	Indicates that a connection is trying to be made in a GSKit environment after the GSK_ENVIRONMENT_ CLOSE_OPTIONS has been set to GSK_DELAYED_ ENVIRONMENT_CLOSE and gsk_environment_close() function has been called.
0x000000c9	201	GSK_NO_KEYFILE_PASSWORD	Neither the password nor the stash-file name was specified, so the key file could not be initialized.

Table 10. GSKit general return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x000000ca	202	GSK_KEYRING_OPEN_ERROR	Unable to open the key file or the Microsoft® Certificate Store. The path was specified incorrectly, or the file permissions did not allow the file to be opened, or the file format is incorrect.
0x000000cb	203	GSK_RSA_TEMP_KEY_PAIR	Unable to generate a temporary key pair. Report this error to service.
0x000000cc	204	GSK_ERROR_LDAP_NO_SUCH_OBJECT	A User Name object was specified that is not found.
0x000000cd	205	GSK_ERROR_LDAP_INVALID_ CREDENTIALS	A Password used for an LDAP query is not correct.
0x000000ce	206	GSK_ERROR_BAD_INDEX	An index into the Fail Over list of LDAP servers was not correct.
0x000000cd	207	GSK_ERROR_FIPS_NOT_SUPPORTED	Attempt to put GSKit into FIPS mode has failed.
0x0000012d	301	GSK_CLOSE_FAILED	Indicates that the GSKit environment close request was not properly handled. This is most likely due to a gsk_secure_socket*() command being attempted after a gsk_close_environment() call.
0x00000191	401	GSK_ERROR_BAD_DATE	The system date was set to an invalid value.
0x00000192	402	GSK_ERROR_NO_CIPHERS	Neither SSLV2 nor SSLV3 is enabled.
0x00000193	403	GSK_ERROR_NO_CERTIFICATE	The required certificate was not received from partner.
0x00000194	404	GSK_ERROR_BAD_CERTIFICATE	The received certificate was formatted incorrectly.
0x00000195	405	GSK_ERROR_UNSUPPORTED_ CERTIFICATE_TYPE	The received certificate type was not supported.
0x00000196	406	GSK_ERROR_IO	An I/O error occurred on a data read or write operation.
0x00000197	407	GSK_ERROR_BAD_KEYFILE_LABEL	The specified label in the key file could not be found.
0x00000198	408	GSK_ERROR_BAD_KEYFILE_ PASSWORD	The specified key file password is incorrect. The key file could not be used. The key file also might be corrupt.
0x00000199	409	GSK_ERROR_BAD_KEY_LEN_ FOR_EXPORT	In a restricted cryptography environment, the key size is too long to be supported.
0x0000019a	410	GSK_ERROR_BAD_MESSAGE	An incorrectly formatted SSL message was received from the partner.

Table 10. GSKit general return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x0000019b	411	GSK_ERROR_BAD_MAC	The message authentication code (MAC) was not successfully verified.
0x0000019c	412	GSK_ERROR_UNSUPPORTED	Unsupported SSL protocol or unsupported certificate type.
0x0000019d	413	GSK_ERROR_BAD_CERT_SIG	The received certificate contained an incorrect signature.
0x0000019e	414	GSK_ERROR_BAD_CERT	Incorrectly formatted certificate received from partner.
0x0000019f	415	GSK_ERROR_BAD_PEER	Invalid SSL protocol received from partner.
0x000001a0	416	GSK_ERROR_PERMISSION_DENIED	Report this internal error to service.
0x000001a1	417	GSK_ERROR_SELF_SIGNED	The self-signed certificate is not valid.
0x000001a2	418	GSK_ERROR_NO_READ_FUNCTION	The read operation failed. Report this internal error to service.
0x000001a3	419	GSK_ERROR_NO_WRITE_FUNCTION	The write operation failed. Report this internal error to service.
0x000001a4	420	GSK_ERROR_SOCKET_CLOSED	The partner closed the socket before the protocol completed.
0x000001a5	421	GSK_ERROR_BAD_V2_CIPHER	The specified V2 cipher is not valid.
0x000001a6	422	GSK_ERROR_BAD_V3_CIPHER	The specified V3 cipher is not valid.
0x000001a7	423	GSK_ERROR_BAD_SEC_TYPE	Report this internal error to service.
0x000001a8	424	GSK_ERROR_BAD_SEC_ TYPE_COMBINATION	Report this internal error to service.
0x000001a9	425	GSK_ERROR_HANDLE_ CREATION_FAILED	The handle could not be created. Report this internal error to service.
0x000001aa	426	GSK_ERROR_INITIALIZATION_ FAILED	Initialization failed. Report this internal error to service.
0x000001ab	427	GSK_ERROR_LDAP_NOT_AVAILABLE	When validating a certificate, unable to access the specified LDAP directory.
0x000001ac	428	GSK_ERROR_NO_PRIVATE_KEY	The specified key did not contain a private key.
0x000001ad	429	GSK_ERROR_PKCS11_ LIBRARY_NOTLOADED	A failed attempt was made to load the specified PKCS11 shared library.
0x000001ae	430	GSK_ERROR_PKCS11_TOKEN_ LABELMISMATCH	The PKCS #11 driver failed to find the token specified by the caller.
0x000001af	431	GSK_ERROR_PKCS11_TOKEN_ NOTPRESENT	A PKCS #11 token is not present in the slot.

Table 10. GSKit general return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x000001b0	432	GSK_ERROR_PKCS11_TOKEN_ BADPASSWORD	The password/pin to access the PKCS #11 token is invalid.
0x000001b1	433	GSK_ERROR_INVALID_V2_HEADER	The SSL header received was not a properly SSLV2 formatted header.
0x000001b2	434	GSK_CSP_OPEN_ERROR	Unable to access the hardware-based cryptographic service provider (CSP). Either the given CSP name is not registered in the system or the specified CSP name is registered but the certificate store failed to open.
0x000001b3	435	GSK_CONFLICTING_ATTRIBUTE_ SETTING	Attribute setting conflict between PKCS11, CMS key database, and Microsoft Crypto API.
0x000001b4	436	GSK_UNSUPPORTED_PLATFORM	The requested function is not supported on the platform that the application is running. For example, the Microsoft Crypto API is not supported on platforms other than Windows 2000.
0x000001b5	437	GSK_ERROR_INCORRECT_ SESSION_TYPE	Incorrect value is returned from the reset session type callback function. Only GSKit GSK_SERVER_SESSION or GSK_SERVER_SESSION_ WITH_CL_AUTH is allowed.
0x000001f5	501	GSK_INVALID_BUFFER_SIZE	The buffer size is negative or zero.
0x000001f6	502	GSK_WOULD_BLOCK	Used with non-blocking I/O.
0x00000259	601	GSK_ERROR_NOT_SSLV3	SSLV3 is required for reset_cipher, and the connection uses SSLV2.
0x0000025a	602	GSK_MISC_INVALID_ID	An invalid ID was specified for the gsk_secure_soc_misc function call.
0x000002bd	701	GSK_ATTRIBUTE_INVALID_ID	The function call has an invalid ID. This also might be caused by specifying an environment handle when a handle for a SSL connection should be used.
0x000002be	702	GSK_ATTRIBUTE_INVALID_LENGTH	The attribute has a negative length, which is invalid.
0x000002bf	703	GSK_ATTRIBUTE_INVALID_ ENUMERATION	The enumeration value is invalid for the specified enumeration type.
0x000002c0	704	GSK_ATTRIBUTE_INVALID_ SID_CACHE	Invalid parameter list for replacing the Session ID (SID) cache routines.

Table 10. GSKit general return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x000002c1	705	GSK_ATTRIBUTE_INVALID_ NUMERIC_VALUE	When setting a numeric attribute, the specified value is invalid for the specific attribute being set.
0x000002c2	706	GSK_CONFLICTING_VALIDATION_ SETTING	Conflicting parameters have been set for additional certificate validation.
0x000002c3	707	GSK_AES_UNSUPPORTED	The cipher specification included an AES cipher that is not supported on the system of execution.
0x000002c4	708	GSK_PEERID_LENGTH_ERROR	The length of the peer ID is incorrect. It must be less than or equal to 16 bytes.
0x000002c5	709	GSK_CIPHER_INVALID_WHEN_ FIPS_MODE_OFF	Given cipher is not allowed when FIPS mode is off.
0x000002c6	710	GSK_CIPHER_INVALID_WHEN_ FIPS_MODE_ON	No FIPS approved cipher have been selected in FIPS mode.
0x00000641	1601	GSK_TRACE_STARTED	The trace started successfully.
0x00000642	1602	GSK_TRACE_STOPPED	The trace stopped successfully.
0x00000643	1603	GSK_TRACE_NOT_STARTED	No trace file was previously started so it cannot be stopped.
0x00000644	1604	GSK_TRACE_ALREADY_STARTED	Trace file already started so it cannot be started again.
0x00000645	1605	GSK_TRACE_OPEN_FAILED	Trace file can not be opened. The first parameter of gsk_start_trace() must be a valid full path file name.

Key management return codes

Table 11. Key management return codes

Return code (hexadecimal)	Return code (decimal)	Constant
0x00000000	0	GSKKM_ERR_OK
0x00000000	0	GSKKM_ERR_SUCCESS
0x00000001	1	GSKKM_ERR_UNKNOWN
0x00000002	2	GSKKM_ERR_ASN
0x00000003	3	GSKKM_ERR_ASN_INITIALIZATION
0x00000004	4	GSKKM_ERR_ASN_PARAMETER
0x00000005	5	GSKKM_ERR_DATABASE
0x00000006	6	GSKKM_ERR_DATABASE_OPEN
0x00000007	7	GSKKM_ERR_DATABASE_RE_OPEN
0x00000008	8	GSKKM_ERR_DATABASE_CREATE
0x00000009	9	GSKKM_ERR_DATABASE_ALREADY_EXISTS

Table 11. Key management return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant
0x0000000a	10	GSKKM_ERR_DATABASE_DELETE
0x0000000b	11	GSKKM_ERR_DATABASE_NOT_OPENED
0x0000000c	12	GSKKM_ERR_DATABASE_READ
0x0000000d	13	GSKKM_ERR_DATABASE_WRITE
0x0000000e	14	GSKKM_ERR_DATABASE_VALIDATION
0x0000000f	15	GSKKM_ERR_DATABASE_INVALID_VERSION
0x00000010	16	GSKKM_ERR_DATABASE_INVALID_PASSWORD
0x00000011	17	GSKKM_ERR_DATABASE_INVALID_FILE_TYPE
0x00000012	18	GSKKM_ERR_DATABASE_CORRUPTION
0x00000013	19	GSKKM_ERR_DATABASE_PASSWORD_ CORRUPTION
0x00000014	20	GSKKM_ERR_DATABASE_KEY_INTEGRITY
0x00000015	21	GSKKM_ERR_DATABASE_DUPLICATE_KEY
0x00000016	22	GSKKM_ERR_DATABASE_DUPLICATE_ KEY_RECORD_ID
0x00000017	23	GSKKM_ERR_DATABASE_DUPLICATE_ KEY_LABEL
0x00000018	24	GSKKM_ERR_DATABASE_DUPLICATE_ KEY_SIGNATURE
0x00000019	25	GSKKM_ERR_DATABASE_DUPLICATE_ KEY_UNSIGNED_CERTIFICATE
0x0000001a	26	GSKKM_ERR_DATABASE_DUPLICATE_KEY_ ISSUER_AND_SERIAL_NUMBER
0x0000001b	27	GSKKM_ERR_DATABASE_DUPLICATE_KEY_ SUBJECT_PUBLIC_KEY_INFO
0x0000001c	28	GSKKM_ERR_DATABASE_DUPLICATE_KEY_ UNSIGNED_CRL
0x0000001d	29	GSKKM_ERR_DATABASE_DUPLICATE_LABEL
0x0000001e	30	GSKKM_ERR_DATABASE_PASSWORD_ ENCRYPTION
0x0000001f	31	GSKKM_ERR_DATABASE_LDAP
0x00000020	32	GSKKM_ERR_CRYPTO
0x00000021	33	GSKKM_ERR_CRYPTO_ENGINE
0x00000022	34	GSKKM_ERR_CRYPTO_ALGORITHM
0x00000023	35	GSKKM_ERR_CRYPTO_SIGN
0x00000024	36	GSKKM_ERR_CRYPTO_VERIFY
0x00000025	37	GSKKM_ERR_CRYPTO_DIGEST
0x00000026	38	GSKKM_ERR_CRYPTO_PARAMETER
0x00000027	39	GSKKM_ERR_CRYPTO_UNSUPPORTED_ ALGORITHM
0x00000028	40	GSKKM_ERR_CRYPTO_INPUT_GREATER_ THAN_MODULUS

Table 11. Key management return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant
0x00000029	41	GSKKM_ERR_CRYPTO_UNSUPPORTED_ MODULUS_SIZE
0x0000002a	42	GSKKM_ERR_VALIDATION
0x0000002b	43	GSKKM_ERR_VALIDATION_KEY
0x0000002c	44	GSKKM_ERR_VALIDATION_DUPLICATE_ EXTENSIONS
0x0000002d	45	GSKKM_ERR_VALIDATION_KEY_WRONG_ VERSION
0x0000002e	46	GSKKM_ERR_VALIDATION_KEY_ EXTENSIONS_REQUIRED
0x0000002f	47	GSKKM_ERR_VALIDATION_KEY_VALIDITY
0x00000030	48	GSKKM_ERR_VALIDATION_KEY_VALIDITY_ PERIOD
0x00000031	49	GSKKM_ERR_VALIDATION_KEY_VALIDITY_ PRIVATE_KEY_USAGE
0x00000032	50	GSKKM_ERR_VALIDATION_KEY_ISSUER_ NOT_FOUND
0x00000033	51	GSKKM_ERR_VALIDATION_KEY_MISSING_ REQUIRED_EXTENSIONS
0x00000034	52	GSKKM_ERR_VALIDATION_KEY_BASIC_ CONSTRAINTS
0x00000035	53	GSKKM_ERR_VALIDATION_KEY_SIGNATURE
0x00000036	54	GSKKM_ERR_VALIDATION_KEY_ROOT_KEY_ NOT_TRUSTED
0x00000037	55	GSKKM_ERR_VALIDATION_KEY_IS_REVOKED
0x00000038	56	GSKKM_ERR_VALIDATION_KEY_AUTHORITY_ KEY_IDENTIFIER
0x00000039	57	GSKKM_ERR_VALIDATION_KEY_PRIVATE_KEY_ USAGE_PERIOD
0x0000003a	58	GSKKM_ERR_VALIDATION_SUBJECT_ ALTERNATIVE_NAME
0x0000003b	59	GSKKM_ERR_VALIDATION_ISSUER_ ALTERNATIVE_NAME
0x0000003c	60	GSKKM_ERR_VALIDATION_KEY_USAGE
0x0000003d	61	GSKKM_ERR_VALIDATION_KEY_ UNKNOWN_CRITICAL_EXTENSION
0x0000003e	62	GSKKM_ERR_VALIDATION_KEY_PAIR
0x0000003f	63	GSKKM_ERR_VALIDATION_CRL
0x00000040	64	GSKKM_ERR_MUTEX
0x00000041	65	GSKKM_ERR_PARAMETER
0x00000042	66	GSKKM_ERR_NULL_PARAMETER
0x00000043	67	GSKKM_ERR_NUMBER_SIZE
0x00000044	68	GSKKM_ERR_OLD_PASSWORD
0x00000045	69	GSKKM_ERR_NEW_PASSWORD

Table 11. Key management return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant
0x00000046	70	GSKKM_ERR_PASSWORD_EXPIRATION_TIME
0x00000047	71	GSKKM_ERR_THREAD
0x00000048	72	GSKKM_ERR_THREAD_CREATE
0x00000049	73	GSKKM_ERR_THREAD_WAIT_FOR_EXIT
0x0000004a	74	GSKKM_ERR_IO
0x0000004b	75	GSKKM_ERR_LOAD
0x0000004c	76	GSKKM_ERR_PKCS11
0x0000004d	77	GSKKM_ERR_NOT_INITIALIZED
0x0000004e	78	GSKKM_ERR_DB_TABLE_CORRUPTED
0x0000004f	79	GSKKM_ERR_MEMORY_ALLOCATE
0x00000050	80	GSKKM_ERR_UNSUPPORTED_OPTION
0x00000051	81	GSKKM_ERR_GET_TIME
0x00000052	82	GSKKM_ERR_CREATE_MUTEX
0x00000053	83	GSKKM_ERR_CMDCAT_OPEN
0x00000054	84	GSKKM_ERR_ERRCAT_OPEN
0x00000055	85	GSKKM_ERR_FILENAME_NULL
0x00000056	86	GSKKM_ERR_FILE_OPEN
0x00000057	87	GSKKM_ERR_FILE_OPEN_TO_READ
0x00000058	88	GSKKM_ERR_FILE_OPEN_TO_WRITE
0x00000059	89	GSKKM_ERR_FILE_OPEN_NOT_EXIST
0x0000005a	90	GSKKM_ERR_FILE_OPEN_NOT_ALLOWED
0x0000005b	91	GSKKM_ERR_FILE_WRITE
0x0000005c	92	GSKKM_ERR_FILE_REMOVE
0x0000005d	93	GSKKM_ERR_BASE64_INVALID_DATA
0x0000005e	94	GSKKM_ERR_BASE64_INVALID_MSGTYPE
0x0000005f	95	GSKKM_ERR_BASE64_ENCODING
0x00000060	96	GSKKM_ERR_BASE64_DECODING
0x00000061	97	GSKKM_ERR_DN_TAG_NULL
0x00000062	98	GSKKM_ERR_DN_CN_NULL
0x00000063	99	GSKKM_ERR_DN_C_NULL
0x00000064	100	GSKKM_ERR_INVALID_DB_HANDLE
0x00000065	101	GSKKM_ERR_KEYDB_NOT_EXIST
0x00000066	102	GSKKM_ERR_KEYPAIRDB_NOT_EXIST
0x00000067	103	GSKKM_ERR_PWDFILE_NOT_EXIST
0x00000068	104	GSKKM_ERR_PASSWORD_CHANGE_MATCH
0x00000069	105	GSKKM_ERR_KEYDB_NULL
0x0000006a	106	GSKKM_ERR_REQKEYDB_NULL
0x0000006b	107	GSKKM_ERR_KEYDB_TRUSTCA_NULL
0x0000006c	108	GSKKM_ERR_REQKEY_FOR_CERT_NULL

Table 11. Key management return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant
0x0000006d	109	GSKKM_ERR_KEYDB_PRIVATE_KEY_NULL
0x0000006e	110	GSKKM_ERR_KEYDB_DEFAULT_KEY_NULL
0x0000006f	111	GSKKM_ERR_KEYREC_PRIVATE_KEY_NULL
0x00000070	112	GSKKM_ERR_KEYREC_CERTIFICATE_NULL
0x00000071	113	GSKKM_ERR_CRLS_NULL
0x00000072	114	GSKKM_ERR_INVALID_KEYDB_NAME
0x00000073	115	GSKKM_ERR_UNDEFINED_KEY_TYPE
0x00000074	116	GSKKM_ERR_INVALID_DN_INPUT
0x00000075	117	GSKKM_ERR_KEY_GET_BY_LABEL
0x00000076	118	GSKKM_ERR_LABEL_LIST_CORRUPT
0x00000077	119	GSKKM_ERR_INVALID_PKCS12_DATA
0x00000078	120	GSKKM_ERR_PKCS12_PWD_CORRUPTION
0x00000079	121	GSKKM_ERR_EXPORT_TYPE
0x0000007a	122	GSKKM_ERR_PBE_ALG_UNSUPPORT
0x0000007b	123	GSKKM_ERR_KYR2KDB
0x0000007c	124	GSKKM_ERR_KDB2KYR
0x0000007d	125	GSKKM_ERR_ISSUING_CERTIFICATE
0x0000007e	126	GSKKM_ERR_FIND_ISSUER_CHAIN
0x0000007f	127	GSKKM_ERR_WEBDB_DATA_BAD_FORMAT
0x00000080	128	GSKKM_ERR_WEBDB_NOTHING_TO_WRITE
0x00000081	129	GSKKM_ERR_EXPIRE_DAYS_TOO_LARGE
0x00000082	130	GSKKM_ERR_PWD_TOO_SHORT
0x00000083	131	GSKKM_ERR_PWD_NO_NUMBER
0x00000084	132	GSKKM_ERR_PWD_NO_CONTROL_KEY
0x00000085	133	GSKKM_ERR_SIGNATURE_ALGORITHM
0x00000086	134	GSKKM_ERR_INVALID_DATABASE_TYPE
0x00000087	135	GSKKM_ERR_SECONDARY_KEYDB_TO_OTHER
0x00000088	136	GSKKM_ERR_NO_SECONDARY_KEYDB
0x00000089	137	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_ LABEL_NOT_EXIST
0x0000008a	138	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_ PASSWORD_REQUIRED
0x0000008b	139	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_ PASSWORD_NOT_REQUIRED
0x0000008c	140	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_ LIBRARY_NOT_LOADED
0x0000008d	141	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_ NOT_SUPPORT
0x0000008e	142	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_ FUNCTION_FAILED
0x0000008f	143	GSKKM_ERR_LDAP_USER_NOT_FOUND

Table 11. Key management return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant
0x00000090	144	GSKKM_ERR_LDAP_INVALID_PASSWORD
0x00000091	145	GSKKM_ERR_LDAP_QUERY_ENTRY_FAILED
0x00000092	146	GSKKM_ERR_INVALID_CERT_CHAIN
0x00000093	147	GSKKM_ERR_CERT_ROOT_NOT_TRUSTED
0x00000094	148	GSKKM_ERR_CERT_REVOKED
0x00000095	149	GSKKM_ERR_CRYPTOGRAPHIC_OBJECT_ FUNCTION_FAILED
0x00000096	150	GSKKM_ERR_NO_AVAILABLE_CRL_ DATASOURCE
0x00000097	151	GSKKM_ERR_NO_TOKEN_PRESENT
0x00000098	152	GSKKM_ERR_FIPS_NOT_SUPPORTED
0x00000099	153	GSKKM_ERR_FIPS_CONFLICT_SETTING
0x0000009a	154	GSKKM_ERR_PASSWORD_STRENGTH_FAILED

Using an access token to acquire users' group information (Windows)

An access token is an object that describes the security context of a process or thread. The information in an access token includes the identity and privileges of the user account associated with the process or thread.

When you log on, the system verifies your password by comparing it with information stored in a security database. If the password is authenticated, the system produces an access token. Every process run on your behalf uses a copy of this access token.

An access token can also be acquired based on cached credentials. After you have been authenticated to the system, your credentials are cached by the operating system. The access token of the last logon can be referenced in the cache when it is not possible to contact the domain controller.

The access token includes information about all of the groups you belong to: local groups and various domain groups (global groups, domain local groups, and universal groups).

Note: Group lookup using client authentication is not supported using a remote connection even though access token support is enabled.

To enable access token support, you must use the db2set command to update the DB2_GRP_LOOKUP registry variable. DB2_GRP_LOOKUP can have up to two parameters, separated by a comma:

- The first parameter is for conventional group lookup and can take the values: " ", "LOCAL", or "DOMAIN".
- The second parameter is for token style group lookup and can take the values: "TOKEN", "TOKENDOMAIN", or "TOKENLOCAL".

If the second parameter (TOKEN, TOKENDOMAIN, or TOKENLOCAL) is specified, it takes precedence over conventional group enumeration. If token group enumeration fails, conventional group lookup occurs, if the first parameter of DB2_GRP_LOOKUP was specified.

The meaning of the values TOKEN, TOKENDOMAIN, and TOKENLOCAL are as follows:

- TOKENLOCAL

  The token is used to enumerate groups at the local machine (this is equivalent to conventional ″LOCAL″ group lookup).
- TOKENDOMAIN

  The token is used to enumerate groups at the location where the user is defined (at local machine for a local user and at the domain for a domain user). This is equivalent to conventional ″ ″, or ″DOMAIN″ group lookup.
- TOKEN

  The token is used to enumerate groups at both the domain and on the local machine. For a local user, the groups returned will contain local groups. For a domain user, the groups returned will contain both domain and local groups. There is no equivalent in conventional group lookup.

For example, the following setting of DB2_GRP_LOOKUP enables access token support for enumerating local groups:

```
db2set DB2_GRP_LOOKUP=LOCAL,TOKENLOCAL
```

The next example enables access token support for enumerating groups at both the local machine as well as the location where the user ID is defined (if the account is defined at the domain):

```
db2set DB2_GRP_LOOKUP=,TOKEN
```

This final example enables access token support for enumerating domain groups at the location where the user ID is defined:

```
db2set DB2_GRP_LOOKUP=DOMAIN,TOKENDOMAIN
```

Note: Access token support can be enabled with all authentications types except CLIENT authentication.

Details on security based on operating system

Each operating system provides ways to manage security. Some of the security issues associated with the operating systems are discussed in this section.

Defining which users hold SYSADM authority (Windows)

Certain users have SYSADM authority if the **sysadm_group** database manager configuration parameter is not set (that is, it is NULL).

These users are:

- Members of the local Administrators group
- Members of the Administrators group at the Domain Controller, if the DB2 database manager is configured to enumerate groups for users at the location where the users are defined (you can use the **DB2_GRP_LOOKUP** environment variable to configure group enumeration)
- Members of the DB2ADMNS group, if Windows extended security is enabled. The location of the DB2ADMNS group is decided during installation.

- The LocalSystem account

There are cases where the above default behavior is not desirable. You can use the **sysadm_group** database manager configuration parameter to override this behavior by using one of the following methods:

- Create a local group on the DB2 server machine and add to it users (domain users or local users) that you want to have SYSADM authority. The DB2 database manager should be configured to enumerate groups for the user on the local machine.
- Create a domain group and add to it the users that you want to have SYSADM authority. The DB2 database manager should be configured to enumerate groups for users at the location where the users are defined.

Then update the **sysadm_group** database manager configuration parameter to this group, using the following commands:

```
db2stop
DB2 UPDATE DBM CFG USING SYSADM_GROUP group_name
db2start
```

DB2 and UNIX security

There are some security considerations specific to UNIX platforms that you need to be aware of.

The DB2 database does not support root acting directly as a database administrator. You should use su - <instance owner> as the database administrator.

For security reasons, in general, do not use the instance name as the Fenced ID. However, if you are not planning to use fenced UDFs or stored procedures, you can set the Fenced ID to the instance name instead of creating another user ID.

The recommendation is to create a user ID that is recognized as being associated with this group. The user for fenced UDFs and stored procedures is specified as a parameter of the instance creation script (db2icrt ... -u <FencedID>). This is not required if you install the DB2 Clients or the DB2 Software Developer's Kit.

Chapter 17. Label-based access control (LBAC)

This section explains what you need to know in order to use label-based access control (LBAC).

Label-based access control (LBAC)

Label-based access control (LBAC) greatly increases the control you have over who can access your data. LBAC lets you decide exactly who has write access and who has read access to individual rows and individual columns.

What LBAC does

The LBAC capability is very configurable and can be tailored to match your particular security environment. All LBAC configuration is performed by a *security administrator*, which is a user that has been granted the SECADM authority.

A security administrator configures the LBAC system by creating security label components. A *security label component* is a database object that represents a criterion you want to use to determine if a user should access a piece of data. For example, the criterion can be whether the user is in a certain department, or whether they are working on a certain project. A *security policy* describes the criteria that will be used to decide who has access to what data. A security policy contains one or more security label components. Only one security policy can be used to protect any one table but different tables can be protected by different security policies.

After creating a security policy, a security administrator creates objects, called *security labels* that are part of that policy. Security labels contain security label components. Exactly what makes up a security label is determined by the security policy and can be configured to represent the criteria that your organization uses to decide who should have access to particular data items. If you decide, for instance, that you want to look at a person's position in the company and what projects they are part of to decide what data they should see, then you can configure your security labels so that each label can include that information. LBAC is flexible enough to let you set up anything from very complicated criteria, to a very simple system where each label represents either a ″high″ or a ″low″ level of trust.

Once created, a security label can be associated with individual columns and rows in a table to protect the data held there. Data that is protected by a security label is called *protected data*. A security administrator allows users access to protected data by granting them security labels. When a user tries to access protected data, that user's security label is compared to the security label protecting the data. The protecting label will block some security labels and not block others.

A user, a role, or a group is allowed to hold security labels for multiple security policies at once. For any given security policy, however, a use, a role, or a group can hold at most one label for read access and one label for write access.

A security administrator can also grant exemptions to users. An *exemption* allows you to access protected data that your security labels might otherwise prevent you from accessing. Together your security labels and exemptions are called your *LBAC credentials*.

If you try to access a protected column that your LBAC credentials do not allow you to access then the access will fail and you will get an error message.

If you try to read protected rows that your LBAC credentials do not allow you to read then DB2 acts as if those rows do not exist. Those rows cannot be selected as part of any SQL statement that you run, including SELECT, UPDATE, or DELETE. Even the aggregate functions ignore rows that your LBAC credentials do not allow you to read. The COUNT(*) function, for example, will return a count only of the rows that you have read access to.

Views and LBAC

You can define a view on a protected table the same way you can define one on a non-protected table. When such a view is accessed the LBAC protection on the underlying table is enforced. The LBAC credentials used are those of the session authorization ID. Two users accessing the same view might see different rows depending on their LBAC credentials.

Referential integrity constraints and LBAC

The following rules explain how LBAC rules are enforced in the presence of referential integrity constraints:

- **Rule 1**: The LBAC read access rules are NOT applied for internally generated scans of child tables. This is to avoid having orphan children.
- **Rule 2**: The LBAC read access rules are NOT applied for internally generated scans of parent tables
- **Rule 3**: The LBAC write rules are applied when a CASCADE operation is performed on child tables. For example, If a user deletes a parent, but cannot delete any of the children because of an LBAC write rule violation, then the delete should be rolled-back and an error raised.

Storage overhead when using LBAC

When you use LBAC to protect a table at the row level, the additional storage cost is the cost of the row security label column. This cost depends on the type of security label chosen. For example, if you create a security policy with two components to protect a table, a security label from that security policy will occupy 16 bytes (8 bytes for each component). Because the row security label column is treated as a not nullable VARCHAR column, the total cost in this case would be 20 bytes per row. In general, the total cost per row is (*N**8 + 4) bytes where *N* is the number of components in the security policy protecting the table.

When you use LBAC to protect a table at the column level, the column security label is meta-data (that is, it is stored together with the column's meta-data in the SYSCOLUMNS catalog table). This meta-data is simply the ID of the security label protecting the column. The user table does not incur any storage overhead in this case.

What LBAC does not do

- LBAC will never allow access to data that is forbidden by discretionary access control.

  Example: If you do not have permission to read from a table then you will not be allowed to read data from that table--even the rows and columns to which LBAC would otherwise allow you access.
- Your LBAC credentials only limit your access to protected data. They have no effect on your access to unprotected data.
- LBAC credentials are not checked when you drop a table or a database, even if the table or database contains protected data.
- LBAC credentials are not checked when you back up your data. If you can run a backup on a table, which rows are backed up is not limited in any way by the LBAC protection on the data. Also, data on the backup media is not protected by LBAC. Only data in the database is protected.
- LBAC cannot be used to protect any of the following types of tables:
  - A materialized query table (MQT)
  - A table that a materialized query table (MQT) depends on
  - A staging table
  - A table that a staging table depends on
  - A typed table
- LBAC protection cannot be applied to a nickname.

LBAC tutorial

A tutorial leading you through the basics of using LBAC is available online at http://www.ibm.com/developerworks/db2 and is called DB2 Label-Based Access Control, a practical guide.

LBAC security policies

The security administrator uses a security policy to define criteria that determine who has write access and who has read access to individual rows and individual columns of tables.

A security policy includes this information:

- What security label components are used in the security labels that are part of the policy
- What rules are used when comparing those security label components
- Which of certain optional behaviors are used when accessing data protected by the policy
- What additional security labels and exemptions are to be considered when enforcing access to data protected by the security policy. For example, the option to consider or not to consider security labels granted to roles and groups is controlled through the security policy.

Every protected table must have one and only one security policy associated with it. Rows and columns in that table can only be protected with security labels that are part of that security policy and all access of protected data follows the rules of that policy. You can have multiple security policies in a single database but you cannot have more than one security policy protecting any given table.

Creating a security policy

You must be a security administrator to create a security policy. You create a security policy with the SQL statement CREATE SECURITY POLICY. The security label components listed in a security policy must be created before the CREATE SECURITY POLICY statement is executed. The order in which the components are listed when a security policy is created does not indicate any sort of precedence or other relationship among the components but it is important to know the order when creating security labels with built-in functions like SECLABEL.

From the security policy you have created, you can create security labels to protect your data.

Altering a security policy

A security administrator can use the ALTER SECURITY POLICY statement to modify a security policy.

Dropping a security policy

You must be a security administrator to drop a security policy. You drop a security policy using the SQL statement DROP.

You cannot drop a security policy if it is associated with (added to) any table.

LBAC security label components

This section explains what you need to know in order to use LBAC security label components.

LBAC security label components overview

A *security label component* is a database object that is part of label-based access control (LBAC). You use security label components to model your organization's security structure.

A security label component can represent any criteria that you might use to decide if a user should have access to a given piece of data. Typical examples of such criteria include:

- How well trusted the user is
- What department the user is in
- Whether the user is involved in a particular project

Example: If you want the department that a user is in to affect which data they can access, you could create a component named dept and define elements for that component that name the various departments in your company. You would then include the component dept in your security policy.

An *element* of a security label component is one particular ″setting″ that is allowed for that component.

Example: A security label component that represents a level of trust might have the four elements: Top Secret, Secret, Classified, and Unclassified.

Creating a security label component

You must be a security administrator to create a security label component. You create security label components with the SQL statement CREATE SECURITY LABEL COMPONENT.

When you create a security label component you must provide:

- A name for the component
- What type of component it is (ARRAY, TREE, or SET)
- A complete list of allowed elements
- For types ARRAY and TREE you must describe how each element fits into the structure of the component

After creating your security label components, you can create a security policy based on these components. From this security policy, you can create security labels to protect your data.

Types of components

There are three types of security label components:

- TREE: Each element represents a node in a tree structure
- ARRAY: Each element represents a point on a linear scale
- SET: Each element represents one member of a set

The types are used to model the different ways in which elements can relate to each other. For example, if you are creating a component to describe one or more departments in a company you would probably want to use a component type of TREE because most business structures are in the form of a tree. If you are creating a component to represent the level of trust that a person has, you would probably use a component of type ARRAY because for any two levels of trust, one will always be higher than the other.

The details of each type, including detailed descriptions of the relationships that the elements can have with each other, are described in their own section.

Altering security label components

The security administrator can use the ALTER SECURITY LABEL COMPONENT statement to modify a security label component.

Dropping a security label component

You must be a security administrator to drop a security label component. You drop a security label component with the SQL statement DROP.

LBAC security label component type: SET

SET is one type of security label component that can be used in a label-based access control (LBAC) security policy.

Components of type SET are unordered lists of elements. The only comparison that can be made for elements of this type of component is whether or not a given element is in the list.

LBAC security label component type: ARRAY

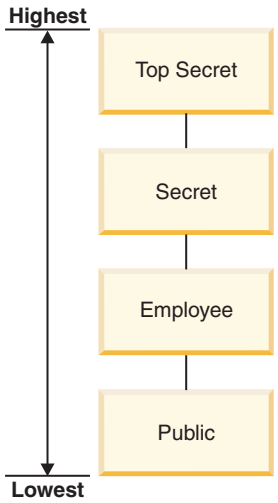
ARRAY is one type of security label component.

In the ARRAY type of component the order in which the elements are listed when the component is created defines a scale with the first element listed being the highest value and the last being the lowest.

Example: If the component mycomp is defined in this way:

```
CREATE SECURITY LABEL COMPONENT mycomp
  ARRAY [ 'Top Secret', 'Secret', 'Employee', 'Public' ]
```

Then the elements are treated as if they are organized in a structure like this:



In a component of type ARRAY, the elements can have these sorts of relationships to each other:

Higher than
Element A is higher than element B if element A is listed earlier in the ARRAY clause than element B.

Lower than
Element A is lower than element B if element A is listed later in the ARRAY clause than element B

LBAC security label component type: TREE

TREE is one type of security label component that can be used in a label-based access control (LBAC) security policy.

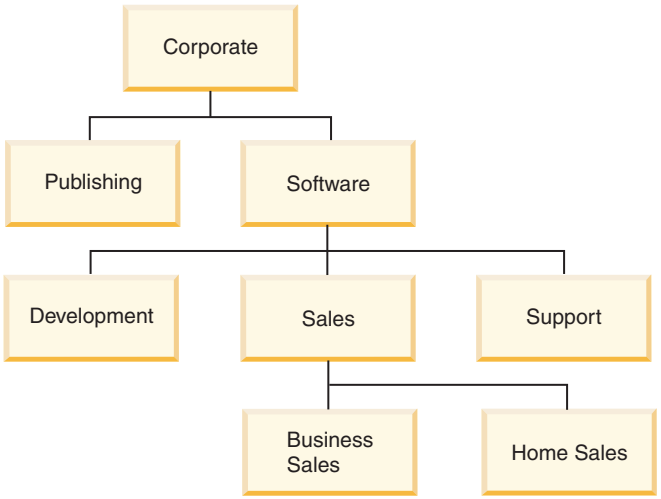
In the TREE type of component the elements are treated as if they are arranged in a tree structure. When you specify an element that is part of a component of type TREE you must also specify which other element it is under. The one exception is the first element which must be specified as being the ROOT of the tree. This allows you to organize the elements in a tree structure.

Example: If the component mycomp is defined this way:

```
CREATE SECURITY LABEL COMPONENT mycomp
TREE (
    'Corporate'      ROOT,
    'Publishing'     UNDER 'Corporate',
```

```
    'Software'       UNDER 'Corporate',
    'Development'    UNDER 'Software',
    'Sales'          UNDER 'Software',
    'Support'        UNDER 'Software'
    'Business Sales' UNDER 'Sales'
    'Home Sales'     UNDER 'Sales'
)
```

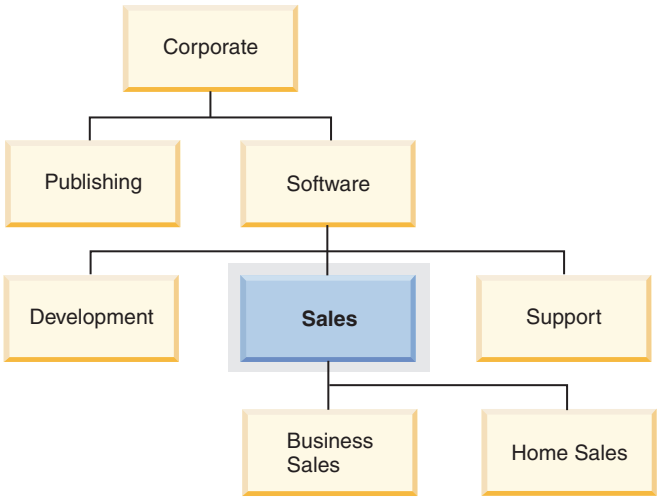
Then the elements are treated as if they are organized in a tree structure like this:



In a component of type TREE, the elements can have these types of relationships to each other:

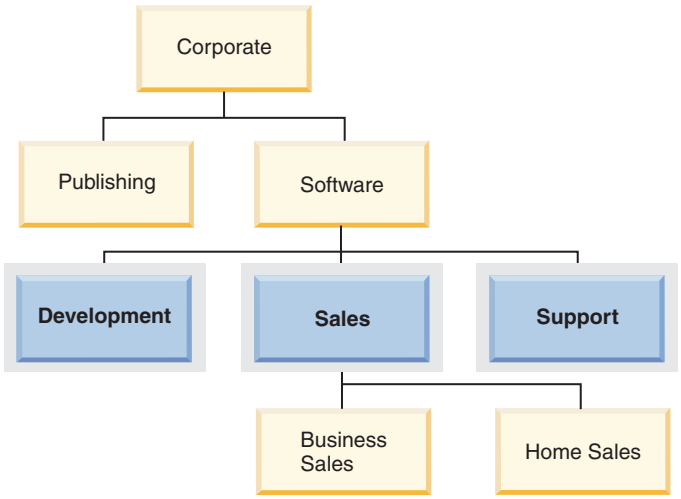
Parent Element A is a parent of element B if element B is UNDER element A.

Example: This diagram shows the parent of the Business Sales element:



Child Element A is a child of element B if element A is UNDER element B.

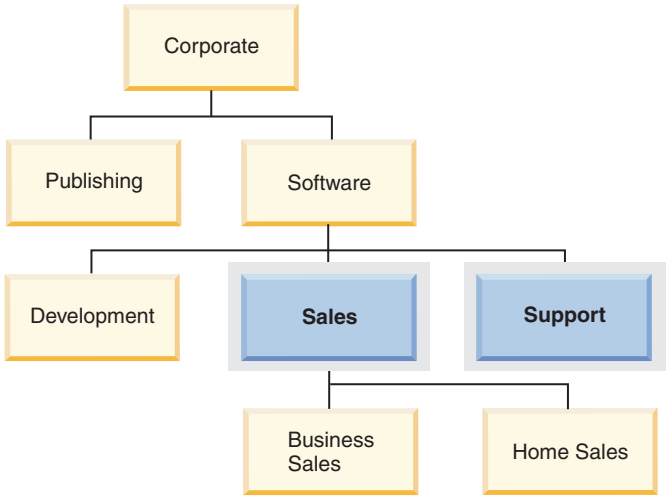
Example: This diagram shows the children of the Software element:



Sibling

Two elements are siblings of each other if they have the same parent.

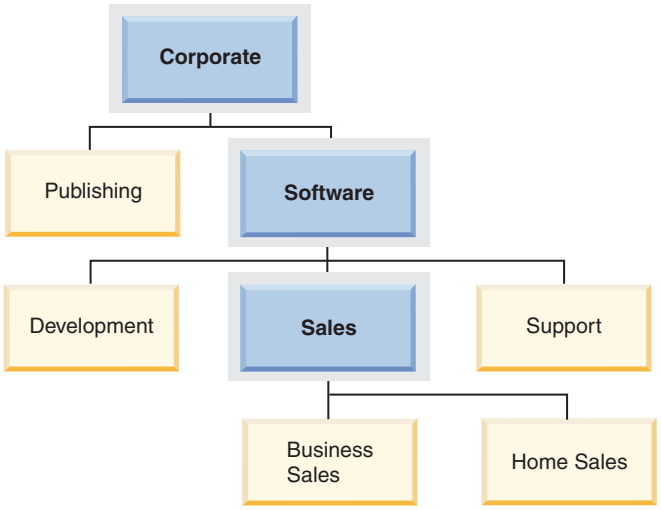
Example: This diagram shows the siblings of the Development element:



Ancestor

Element A is an ancestor of element B if it is the parent of B, or if it is the parent of the parent of B, and so on. The root element is an ancestor of all other elements in the tree.

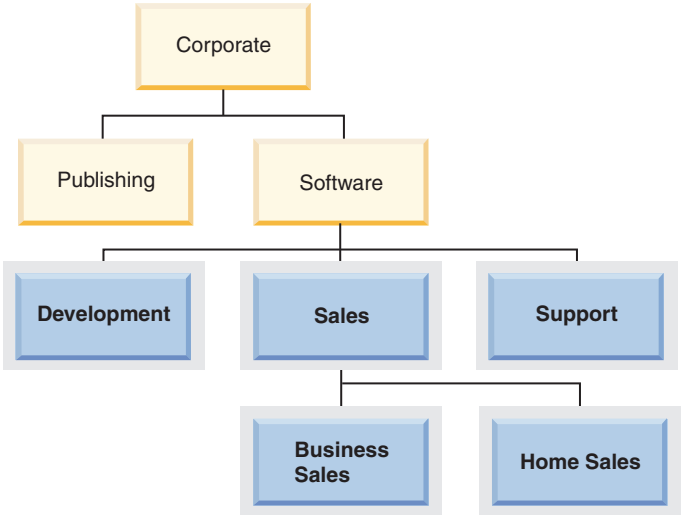
Example: This diagram shows the ancestors of the Home Sales element:



Descendent

Element A is a descendent of element B if it is the child of B, or if it is the child of a child of B, and so on.

Example: This diagram shows the descendents of the Software element:



LBAC security labels

In label-based access control (LBAC) a *security label* is a database object that describes a certain set of security criteria. Security labels are applied to data in order to protect the data. They are granted to users to allow them to access protected data.

When a user tries to access protected data, their security label is compared to the security label that is protecting the data. The protecting security label will block some security labels and not block others. If a user's security label is blocked then the user cannot access the data.

Every security label is part of exactly one security policy and includes one value for each component in that security policy. A *value* in the context of a security label component is a list of zero or more of the elements allowed by that component.

Values for ARRAY type components can contain zero or one element, values for other types can have zero or more elements. A value that does not include any elements is called an *empty value*.

Example: If a TREE type component has the three elements Human Resources, Sales, and Shipping then these are some of the valid values for that component:

- Human Resources (or any of the elements by itself)
- Human Resources, Shipping (or any other combination of the elements as long as no element is included more than once)
- *An empty value*

Whether a particular security label will block another is determined by the values of each component in the labels and the LBAC rule set that is specified in the security policy of the table. The details of how the comparison is made are given in the topic that discusses how LBAC security labels are compared.

When security labels are converted to a text string they use the format described in the topic that discusses the format for security label values.

Creating security labels

You must be a security administrator to create a security label. You create a security label with the SQL statement CREATE SECURITY LABEL. When you create a security label you provide:

- A name for the label
- The security policy that the label is part of
- Values for one or more of the components included in the security policy

Any components for which a value is not specified is assumed to have an empty value. A security label must have at least one non-empty value.

Altering security labels

Security labels cannot be altered. The only way to change a security label is to drop it and re-create it. However, the *components* of a security label can be modified by a security administrator (using the ALTER SECURITY LABEL COMPONENT statement).

Dropping security labels

You must be a security administrator to drop a security label. You drop a security label with the SQL statement DROP. You cannot drop a security label that is being used to protect data anywhere in the database or that is currently held by one or more users.

Granting security labels

You must be a security administrator to grant a security label to a user, a group, or a role. You grant a security label with the SQL statement GRANT SECURITY LABEL. When you grant a security label you can grant it for read access, for write access, or for both read and write access. A user, a group, or a role cannot hold more than one security label from the same security policy for the same type of access.

Revoking security labels

You must be a security administrator to revoke a security label from a user, group, or role. To revoke a security label, use the SQL statement REVOKE SECURITY LABEL.

Data types compatible with security labels

Security labels have a data type of SYSPROC.DB2SECURITYLABEL. Data conversion is supported between SYSPROC.DB2SECURITYLABEL and VARCHAR(128) FOR BIT DATA.

Determining the security labels held by users

You can use the following query to determine the security labels that are held by users:

```
SELECT A.grantee, B.secpolicyname, c.seclabelname
FROM syscat.securitylabelaccess A, syscat.securitypolicies B, syscat.securitylabels C
WHERE A.seclabelid = C.seclabelid and B.secpolicyid = C.secpolicyid
```

Format for security label values

Sometimes the values in a security label are represented in the form of a character string, for example when using the built-in function SECLABEL.

When the values in a security label are represented as a string, they are in the following format:

- The values of the components are listed from left to right in the same order that the components are listed in the CREATE SECURITY POLICY statement for the security policy
- An element is represented by the name of that element
- Elements for different components are separated by a colon (:)
- If more than one element are given for the same component the elements are enclosed in parentheses (()) and are separated by a comma (,)
- Empty values are represented by a set of empty parentheses (())

Example: A security label is part of a security policy that has these three components in this order: Level, Department, and Projects. The security label has these values:

Table 12. Example values for a security label

Component	Values
Level	Secret
Department	*Empty value*
Projects	• Epsilon 37 • Megaphone • Cloverleaf

This security label values look like this as a string:

```
'Secret:():(Epsilon 37,Megaphone,Cloverleaf)'
```

How LBAC security labels are compared

When you try to access data protected by label-based access control (LBAC), your LBAC credentials are compared to one or more security labels to see if the access is blocked. Your LBAC credentials are any security labels you hold plus any exemptions that you hold.

There are only two types of comparison that can be made. Your LBAC credentials can be compared to a single security label for read access or your LBAC credentials compared to a single security label for write access. Updating and deleting are treated as being a read followed by a write. When an operation requires multiple comparisons to be made, each is made separately.

Which of your security labels is used

Even though you might hold multiple security labels only one is compared to the protecting security label. The label used is the one that meets these criteria:

- It is part of the security policy that is protecting the table being accessed.
- It was granted for the type of access (read or write).

If you do not have a security label that meets these criteria then a default security label is assumed that has empty values for all components.

How the comparison is made

Security labels are compared component by component. If a security label does not have a value for one of the components then an empty value is assumed. As each component is examined, the appropriate rules of the LBAC rule set are used to decide if the elements in your value for that component should be blocked by the elements in the value for the same component in the protecting label. If any of your values are blocked then your LBAC credentials are blocked by the protecting security label.

The LBAC rule set used in the comparison is designated in the security policy. To find out what the rules are and when each one is used, see the description of that rule set.

How exemptions affect comparisons

If you hold an exemption for the rule that is being used to compare two values then that comparison is not done and the protecting value is assumed not to block the value in your security label.

Example: The LBAC rule set is DB2LBACRULES and the security policy has two components. One component is of type ARRAY and the other is of type TREE. The user has been granted an exemption on the rule DB2LBACREADTREE, which is the rule used for read access when comparing values of components of type TREE. If the user attempts to read protected data then whatever value the user has for the TREE component, even if it is an empty value, will not block access because that rule is not used. Whether the user can read the data depends entirely on the values of the ARRAY component of the labels.

LBAC security label components

This section explains what you need to know in order to use LBAC security label components.

LBAC rule sets overview

An LBAC rule set is a predefined set of rules that are used when comparing security labels. When the values of a two security labels are being compared, one or more of the rules in the rule set will be used to determine if one value blocks another.

Each LBAC rule set is identified by a unique name. When you create a security policy you must specify the LBAC rule set that will be used with that policy. Any comparison of security labels that are part of that policy will use that LBAC rule set.

Each rule in a rule set is also identified by a unique name. You use the name of a rule when you are granting an exemption on that rule.

How many rules are in a set and when each rule is used can vary from rule set to rule set.

There is currently only one supported LBAC rule set. The name of that rule set is DB2LBACRULES.

LBAC rule set: DB2LBACRULES

The DB2LBACRULES LBAC rule set provides a traditional set of rules for comparing the values of security label components. It protects from both write-up and write-down.

What are write-up and write down?

Write-up and write-down apply only to components of type ARRAY and only to write access. Write up occurs when the value protecting data that you are writing to is higher than your value. Write-down is when the value protecting the data is lower than yours. By default neither write-up nor write-down is allowed, meaning that you can only write data that is protected by the same value that you have.

When comparing two values for the same component, which rules are used depends on the type of the component (ARRAY, SET, or TREE) and what type of access is being attempted (read, or write). This table lists the rules, tells when each is used, and describes how the rule determines if access is blocked.

Table 13. Summary of the DB2LBACRULES rules

Rule name	Used when comparing the values of this type of component	Used when attempting this type of access	Access is blocked when this condition is met
DB2LBACREADARRAY	ARRAY	Read	The user's value is lower than the protecting value.
DB2LBACREADSET	SET	Read	There are one or more protecting values that the user does not hold.

Table 13. Summary of the DB2LBACRULES rules (continued)

Rule name	Used when comparing the values of this type of component	Used when attempting this type of access	Access is blocked when this condition is met
DB2LBACREADTREE	TREE	Read	None of the user's values is equal to or an ancestor of one of the protecting values.
DB2LBACWRITEARRAY	ARRAY	Write	The user's value is higher than the protecting value or lower than the protecting value.[1]
DB2LBACWRITESET	SET	Write	There are one or more protecting values that the user does not hold.
DB2LBACWRITETREE	TREE	Write	None of the user's values is equal to or an ancestor of one of the protecting values.

Note:

1. The DB2LBACWRITEARRAY rule can be thought of as being two different rules combined. One prevents writing to data that is higher than your level (write-up) and the other prevents writing to data that is lower than your level (write-down). When granting an exemption to this rule you can exempt the user from either of these rules or from both.

How the rules handle empty values

All rules treat empty values the same way. An empty value blocks no other values and is blocked by any non-empty value.

DB2LBACREADSET and DB2LBACWRITESET examples

These examples are valid for a user trying to read or trying to write protected data. They assume that the values are for a component of type SET that has these elements: one two three four

Table 14. Examples of applying the DB2LBACREADSET and DB2LBACWRITESET rules.

User's value	Protecting value	Access blocked?
'one'	'one'	Not blocked. The values are the same.
'(one,two,three)'	'one'	Not blocked. The user's value contains the element 'one'.
'(one,two)'	'(one,two,four)'	Blocked. The element 'four' is in the protecting value but not in the user's value.
'()'	'one'	Blocked. An empty value is blocked by any non-empty value.
'one'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

DB2LBACREADTREE and DB2LBACWRITETREE

These examples are valid for both read access and write access. They assume that the values are for a component of type TREE that was defined in this way:

```
CREATE SECURITY LABEL COMPONENT mycomp
TREE (
    'Corporate'      ROOT,
    'Publishing'     UNDER 'Corporate',
    'Software'       UNDER 'Corporate',
    'Development'    UNDER 'Software',
    'Sales'          UNDER 'Software',
    'Support'        UNDER 'Software'
    'Business Sales' UNDER 'Sales'
    'Home Sales'     UNDER 'Sales'
)
```

This means the elements are in this arrangement:

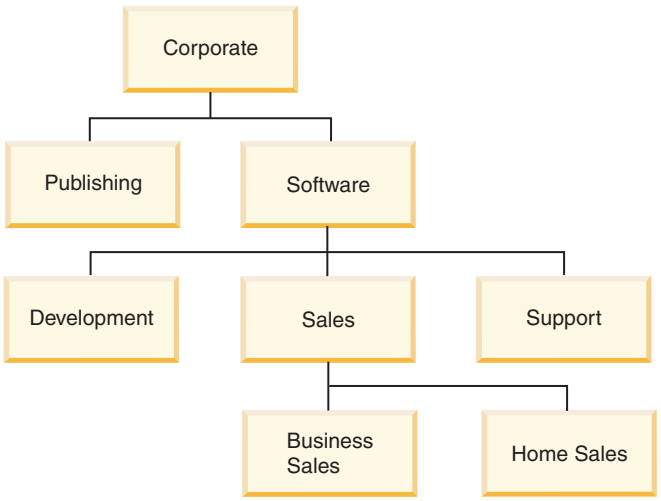


Table 15. Examples of applying the DB2LBACREADTREE and DB2LBACWRITETREE rules.

User's value	Protecting value	Access blocked?
'(Support,Sales)'	'Development'	Blocked. The element 'Development' is not one of the user's values and neither 'Support' nor 'Sales' is an ancestor of 'Development'.
'(Development,Software)'	'(Business Sales,Publishing)'	Not blocked. The element 'Software' is an ancestor of 'Business Sales'.
'(Publishing,Sales)'	'(Publishing,Support)'	Not blocked. The element 'Publishing' is in both sets of values.
'Corporate'	'Development'	Not blocked. The root value is an ancestor of all other values.
'()'	'Sales'	Blocked. An empty value is blocked by any non-empty value.
'Home Sales'	'()'	Not blocked. No value is blocked by an empty value.

Table 15. Examples of applying the DB2LBACREADTREE and DB2LBACWRITETREE rules. (continued)

User's value	Protecting value	Access blocked?
'()'	'()'	Not blocked. No value is blocked by an empty value.

DB2LBACREADARRAY examples

These examples are for read access only. They assume that the values are for a component of type ARRAY that includes these elements in this arrangement:

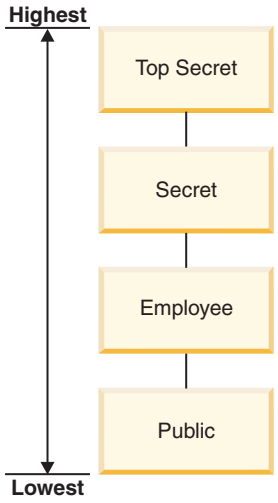


Table 16. Examples of applying the DB2LBACREADARRAY rule.

User's value	Protecting value	Read access blocked?
'Secret'	'Employee'	Not blocked. The element 'Secret' is higher than the element 'Employee'.
'Secret'	'Secret'	Not blocked. The values are the same.
'Secret'	'Top Secret'	Blocked. The element 'Top Secret' is higher than the element 'Secret'.
'()'	'Public'	Blocked. An empty value is blocked by any non-empty value.
'Public'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

DB2LBACWRITEARRAY examples

These examples are for write access only. They assume that the values are for a component of type ARRAY that includes these elements in this arrangement:

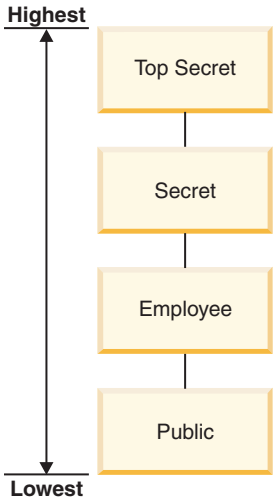


Table 17. Examples of applying the DB2LBACWRITEARRAY rule.

User's value	Protecting value	Write access blocked?
'Secret'	'Employee'	Blocked. The element 'Employee' is lower than the element 'Secret'.
'Secret'	'Secret'	Not blocked. The values are the same.
'Secret'	'Top Secret'	Blocked. The element 'Top Secret' is higher than the element 'Secret'.
'()'	'Public'	Blocked. An empty value is blocked by any non-empty value.
'Public'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

LBAC rule exemptions

When you hold an LBAC rule exemption on a particular rule of a particular security policy, that rule is not enforced when you try to access data protected by that security policy.

An exemption has no effect when comparing security labels of any security policy other than the one for which it was granted.

Example:

There are two tables: T1 and T2. T1 is protected by security policy P1 and T2 is protected by security policy P2. Both security policies have one component. The component of each is of type ARRAY. T1 and T2 each contain only one row of data. The security label that you hold for read access under security policy P1 does not allow you access to the row in T1. The security label that you hold for read access under security policy P2 does not allow you read access to the row in T2.

Now you are granted an exemption on DB2LBACREADARRAY under P1. You can now read the row from T1 but not the row from T2 because T2 is protected by a different security policy and you do not hold an exemption to the DB2LBACREADARRAY rule in that policy.

You can hold multiple exemptions. If you hold an exemption to every rule used by a security policy then you will have complete access to all data protected by that security policy.

Granting LBAC rule exemptions

You must be a security administrator to grant an LBAC rule exemption. To grant an LBAC rule exemption, use the SQL statement GRANT EXEMPTION ON RULE.

When you grant an LBAC rule exemption you provide this information:

- The rule or rules that the exemption is for
- The security policy that the exemption is for
- The user, group, or role to which you are granting the exemption

Important: LBAC rule exemptions provide very powerful access. Do not grant them without careful consideration.

Revoking LBAC rule exemptions

You must be a security administrator to revoke an LBAC rule exemption. To revoke an LBAC rule exemption, use the SQL statement REVOKE EXEMPTION ON RULE.

Determining the rule exemptions held by users

You can use the following query to determine the rule exemptions that are held by users:

```
SELECT A.grantee, A.accessrulename, B.secpolicyname
FROM syscat.securitypolicyexemptions A, syscat.securitypolicies B
WHERE A.secpolicyid = B.secpolicyid
```

Built-in functions for managing LBAC security labels

The built-in functions SECLABEL, SECLABEL_BY_NAME, and SECLABEL_TO_CHAR are provided for managing label-based access control (LBAC) security labels.

Each is described briefly here and in detail in the *SQL Reference*

SECLABEL

This built-in function is used to build a security label by specifying a security policy and values for each of the components in the label. The returned value has a data type of DB2SECURITYLABEL and is a security label that is part of the indicated security policy and has the indicated values for the components. It is not necessary that a security label with the indicated values already exists.

Example: Table T1 has two columns, the first has a data type of DB2SECURITYLABEL and the second has a data type of INTEGER. T1 is protected by security policy P1, which has three security label components: level, departments, and groups. If UNCLASSIFIED is an element of the component level, ALPHA and SIGMA are both elements of the component departments, and G2 is an element of the component groups then a security label could be inserted like this:

```
INSERT INTO T1 VALUES
   ( SECLABEL( 'P1', 'UNCLASSIFIED:(ALPHA,SIGMA):G2' ), 22 )
```

SECLABEL_BY_NAME

This built-in function accepts the name of a security policy and the name of a security label that is part of that security policy. It then returns the indicated security label as a DB2SECURITYLABEL. You must use this function when inserting an existing security label into a column that has a data type of DB2SECURITYLABEL.

Example: Table T1 has two columns, the first has a data type of DB2SECURITYLABEL and the second has a data type of INTEGER. The security label named L1 is part of security policy P1. This SQL inserts the security label:

```
INSERT INTO T1 VALUES ( SECLABEL_BY_NAME( 'P1', 'L1' ), 22 )
```

This SQL statement does not work:

```
INSERT INTO T1 VALUES ( P1.L1, 22 )     // Syntax Error!
```

SECLABEL_TO_CHAR

This built-in function returns a string representation of the values that make up a security label.

Example: Column C1 in table T1 has a data type of DB2SECURITYLABEL. T1 is protected by security policy P1, which has three security label components: level, departments, and groups. There is one row in T1 and the value in column C1 that has these elements for each of the components:

Component	Elements
level	SECRET
departments	DELTA and SIGMA
groups	G3

A user that has LBAC credentials that allow reading the row executes this SQL statement:

```
SELECT SECLABEL_TO_CHAR( 'P1', C1 ) AS C1 FROM T1
```

The output looks like this:

```
C1

'SECRET:(DELTA,SIGMA):G3'
```

Protection of data using LBAC

Label-based access control (LBAC) can be used to protect rows of data, columns of data, or both. Data in a table can only be protected by security labels that are part of the security policy protecting the table. Data protection, including adding a security policy, can be done when creating the table or later by altering the table.

You can add a security policy to a table and protect data in that table as part of the same CREATE TABLE or ALTER TABLE statement.

As a general rule you are not allowed to protect data in such a way that your current LBAC credentials do not allow you to write to that data.

Adding a security policy to a table

You can add a security policy to a table when you create the table by using the SECURITY POLICY clause of the CREATE TABLE statement. You can add a security policy to an existing table by using the ADD SECURITY POLICY clause of the ALTER TABLE statement. You do not need to have SECADM authority or have LBAC credentials to add a security policy to a table.

Security policies cannot be added to types of tables that cannot be protected by LBAC. See the overview of LBAC for a list of table types that cannot be protected by LBAC.

No more than one security policy can be added to any table.

Protecting rows

You can allow protected rows in a new table by including a column with a data type of DB2SECURITYLABEL when you create the table. The CREATE TABLE statement must also add a security policy to the table. You do not need to have SECADM authority or have any LBAC credentials to create such a table.

You can allow protected rows in an existing table by adding a column that has a data type of DB2SECURITYLABEL. To add such a column, either the table must already be protected by a security policy or the ALTER TABLE statement that adds the column must also add a security policy to the table. When the column is added, the security label you hold for write access is used to protect all existing rows. If you do not hold a security label for write access that is part of the security policy protecting the table then you cannot add a column that has a data type of DB2SECURITYLABEL.

After a table has a column of type DB2SECURITYLABEL you protect each new row of data by storing a security label in that column. The details of how this works are described in the topics about inserting and updating LBAC protected data. You must have LBAC credentials to insert rows into a table that has a column of type DB2SECURITYLABEL.

A column that has a data type of DB2SECURITYLABEL cannot be dropped and cannot be changed to any other data type.

Protecting columns

You can protect a column when you create the table by using the SECURED WITH column option of the CREATE TABLE statement. You can add protection to an existing column by using the SECURED WITH option in an ALTER TABLE statement.

To protect a column with a particular security label you must have LBAC credentials that allow you to write to data protected by that security label. You do not have to have SECADM authority.

Columns can only be protected by security labels that are part of the security policy protecting the table. You cannot protect columns in a table that has no

security policy. You are allowed to protect a table with a security policy and protect one or more columns in the same statement.

You can protect any number of the columns in a table but a column can be protected by no more than one security label.

Reading of LBAC protected data

When you try to read data protected by label-based access control (LBAC), your LBAC credentials for reading are compared to the security label that is protecting the data. If the protecting label does not block your credentials you are allowed to read the data.

In the case of a protected column the protecting security label is defined in the schema of the table. The protecting security label for that column is the same for every row in the table. In the case of a protected row the protecting security label is stored in the row in a column of type DB2SECURITYLABEL. It can be different for every row in the table.

The details of how your LBAC credentials are compared to a security label are given in the topic about how LBAC security labels are compared.

Reading protected columns

When you try to read from a protected column your LBAC credentials are compared with the security label protecting the column. Based on this comparison access will either be blocked or allowed. If access is blocked then an error is returned and the statement fails. Otherwise, the statement proceeds as usual.

Trying to read a column that your LBAC credentials do not allow you to read, causes the entire statement to fail.

Example:

Table T1 has two protected columns. The column C1 is protected by the security label L1. The column C2 is protected by the security label L2.

Assume that user Jyoti has LBAC credentials for reading that allow access to security label L1 but not to L2. If Jyoti issues the following SQL statement, the statement will fail:

```
SELECT * FROM T1
```

The statement fails because column C2 is included in the SELECT clause as part of the wildcard (*).

If Jyoti issues the following SQL statement it will succeed:

```
SELECT C1 FROM T1
```

The only protected column in the SELECT clause is C1, and Jyoti's LBAC credentials allow her to read that column.

Reading protected rows

If you do not have LBAC credentials that allow you to read a row it is as if that row does not exist for you.

When you read protected rows, only those rows to which your LBAC credentials allow read access are returned. This is true even if the column of type DB2SECURITYLABEL is not part of the SELECT clause.

Depending on their LBAC credentials, different users might see different rows in a table that has protected rows. For example, two users executing the statement `SELECT COUNT(*) FROM T1` may get different results if T1 has protected rows and the users have different LBAC credentials.

Your LBAC credentials affect not only SELECT statements but also other SQL statements like UPDATE, and DELETE. If you do not have LBAC credentials that allow you to read a row, you cannot affect that row.

Example:

Table T1 has these rows and columns. The column ROWSECURITYLABEL has a data type of DB2SECURITYLABEL.

Table 18. Example values in table T1

LASTNAME	DEPTNO	ROWSECURITYLABEL
Rjaibi	55	L2
Miller	77	L1
Fielding	11	L3
Bird	55	L2

Assume that user Dan has LBAC credentials that allow him to read data that is protected by security label L1 but not data protected by L2 or L3.

Dan issues the following SQL statement:

```
SELECT * FROM T1
```

The SELECT statement returns only the row for Miller. No error messages or warning are returned.

Dan's view of table T1 is this:

Table 19. Example values in view of table T1

LASTNAME	DEPTNO	ROWSECURITYLABEL
Miller	77	L1

The rows for Rjaibi, Fielding, and Bird are not returned because read access is blocked by their security labels. Dan cannot delete or update these rows. They will also not be included in any aggregate functions. For Dan it is as if those rows do not exist.

Dan issues this SQL statement:

```
SELECT COUNT(*) FROM T1
```

The statement returns a value of 1 because only the row for Miller can be read by the user Dan.

Reading protected rows that contain protected columns

Column access is checked before row access. If your LBAC credentials for read access are blocked by the security label protecting one of the columns you are selecting then the entire statement fails. If not, the statement continues and only the rows protected by security labels to which your LBAC credentials allow read access are returned.

Example

The column LASTNAME of table T1 is protected with the security label L1. The column DEPTNO is protected with security label L2. The column ROWSECURITYLABEL has a data type of DB2SECURITYLABEL. T1, including the data, looks like this:

Table 20. Example values in table T1

LASTNAME *Protected by L1*	**DEPTNO** *Protected by L2*	**ROWSECURITYLABEL**
Rjaibi	55	L2
Miller	77	L1
Fielding	11	L3

Assume that user Sakari has LBAC credentials that allow reading data protected by security label L1 but not L2 or L3.

Sakari issues this SQL statement:

```
SELECT * FROM T1
```

The statement fails because the SELECT clause uses the wildcard (*) which includes the column DEPTNO. The column DEPTNO is protected by security label L2, which Sakari's LBAC credentials do not allow her to read.

Sakari next issues this SQL statement:

```
SELECT LASTNAME, ROWSECURITYLABEL FROM T1
```

The select clause does not include any columns that Sakari is not able to read so the statement continues. Only one row is returned, however, because each of the other rows is protected by security label L2 or L3.

Table 21. Example output from query on table T1

LASTNAME	**ROWSECURITYLABEL**
Miller	L1

Inserting of LBAC protected data

When you try to insert data into a protected column, or to insert a new row into a table with protected rows, your LBAC credentials determine how that INSERT statement is handled.

Inserting to protected columns

When you try to insert data into a protected column your LBAC credentials for writing are compared with the security label protecting that column. Based on this comparison access will either be blocked or allowed.

The details of how two security labels are compared are given in the topic about how LBAC security labels are compared.

If access is allowed, the statement proceeds as usual. If access is blocked, then the insert fails and an error is returned.

If you are inserting a row but do not provide a value for a protected column then a default value is inserted if one is available. This happens even if your LBAC credentials do not allow write access to that column. A default is available in the following cases:

- The column was declared with the WITH DEFAULT option
- The column is a generated column
- The column has a default value that is given through a BEFORE trigger
- The column has a data type of DB2SECURITYLABEL, in which case security label that you hold for write access is the default value

Inserting to protected rows

When you insert a new row into a table with protected rows, you do not have to provide a value for the column that is of type DB2SECURITYLABEL. If you do not provide a value for that column, the column is automatically populated with the security label you have been granted for write access. If you have not been granted a security label for write access, an error is returned and the insert fails.

By using built-in functions like SECLABEL, you can explicitly provide a security label to be inserted in a column of type DB2SECURITYLABEL. The provided security label is only used, however, if your LBAC credentials would allow you to write to data that is protected with the security label you are trying to insert.

If you provide a security label that you would not be able to write, then what happens depends on the security policy that is protecting the table. If the security policy has the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option, then the insert fails and an error is returned. If the security policy does not have the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option or if it instead has the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option, then the security label you provide is ignored and if you hold a security label for write access, it is used instead. If you do not hold a security label for write access, an error is returned.

Examples

Table T1 is protected by a security policy named P1 that was created without the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option. Table T1 has two columns but no rows. The columns are LASTNAME and LABEL. The column LABEL has a data type of DB2SECURITYLABEL.

User Joe holds a security label L2 for write access. Assume that the security label L2 allows him to write to data protected by security label L2 but not to data protected by security labels L1 or L3.

Joe issues the following SQL statement:

```
INSERT INTO T1 (LASTNAME, DEPTNO) VALUES ('Rjaibi', 11)
```

Because no security label was included in the INSERT statement, Joe's security label for write access is inserted into the LABEL row.

Table T1 now looks like this:

Table 22. Values in the example table T1 after first INSERT statement

LASTNAME	LABEL
Rjaibi	L2

Joe issues the following SQL statement, in which he explicitly provides the security label to be inserted into the column LABEL:

```
INSERT INTO T1 VALUES ('Miller', SECLABEL_BY_NAME('P1', 'L1') )
```

The SECLABEL_BY_NAME function in the statement returns a security label that is part of security policy P1 and is named L1. Joe is not allowed to write to data that is protected with L1 so he is not allowed to insert L1 into the column LABEL.

Because the security policy protecting T1 was created without the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option the security label that Joe holds for writing is inserted instead. No error or message is returned.

The table now looks like this:

Table 23. Values in example table T1 after second INSERT statement

LASTNAME	LABEL
Rjaibi	L2
Miller	L2

If the security policy protecting the table had been created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option then the insert would have failed and an error would have been returned.

Next Joe is granted an exemption to one of the LBAC rules. Assume that his new LBAC credentials allow him to write to data that is protected with security labels L1 and L2. The security label granted to Joe for write access does not change, it is still L2.

Joe issues the following SQL statement:

```
INSERT INTO T1 VALUES ('Bird', SECLABEL_BY_NAME('P1', 'L1') )
```

Because of his new LBAC credentials Joe is able to write to data that is protected by the security label L1. The insertion of L1 is therefore allowed. The table now looks like this:

Table 24. Values in example table T1 after third INSERT statement

LASTNAME	LABEL
Rjaibi	L2
Miller	L2
Bird	L1

Updating of LBAC protected data

Your LBAC credentials must allow you write access to data before you can update it. In the case of updating a protected row, your LBAC credentials must also allow read access to the row.

Updating protected columns

When you try to update data in a protected column, your LBAC credentials are compared to the security label protecting the column. The comparison made is for write access. If write access is blocked then an error is returned and the statement fails, otherwise the update continues.

The details of how your LBAC credentials are compared to a security label are given in the topic about how LBAC security labels are compared.

Example:

Assume there is a table T1 in which column DEPTNO is protected by a security label L2 and column PAYSCALE is protected by a security label L3. T1, including its data, looks like this:

Table 25. Table T1

EMPNO	**LASTNAME**	**DEPTNO** *Protected by* **L2**	**PAYSCALE** *Protected by* **L3**
1	Rjaibi	11	4
2	Miller	11	7
3	Bird	11	9

User Lhakpa has no LBAC credentials. He issues this SQL statement:

```
UPDATE T1 SET EMPNO = 4
   WHERE LASTNAME = "Bird"
```

This statement executes without error because it does not update any protected columns. T1 now looks like this:

Table 26. Table T1 After Update

EMPNO	**LASTNAME**	**DEPTNO** *Protected by* **L2**	**PAYSCALE** *Protected by* **L3**
1	Rjaibi	11	4
2	Miller	11	7
4	Bird	11	9

Lhakpa next issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 55
   WHERE LASTNAME = "Miller"
```

This statement fails and an error is returned because DEPTNO is protected and Lhakpa has no LBAC credentials.

Assume Lhakpa is granted LBAC credentials and that allow the access summarized in the following table. The details of what those credentials are and what elements are in the security labels are not important for this example.

Security label protecting the data	Can read?	Can Write?
L2	No	Yes
L3	No	No

Lhakpa issues this SQL statement again:

```
UPDATE T1 SET DEPTNO = 55
   WHERE LASTNAME = "Miller"
```

This time the statement executes without error because Lhakpa's LBAC credentials allow him to write to data protected by the security label that is protecting the column DEPTNO. It does not matter that he is not able to read from that same column. The data in T1 now looks like this:

Table 27. Table T1 After Second Update

EMPNO	**LASTNAME**	**DEPTNO** *Protected by* **L2**	**PAYSCALE** *Protected by* **L3**
1	Rjaibi	11	4
2	Miller	55	7
4	Bird	11	9

Next Lhakpa issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 55, PAYSCALE = 4
   WHERE LASTNAME = "Bird"
```

The column PAYSCALE is protected by the security label L3 and Lhakpa's LBAC credentials do not allow him to write to it. Because Lhakpa is unable to write to the column, the update fails and no data is changed.

Updating protected rows

If your LBAC credentials do not allow you to read a row, then it is as if that row does not exist for you so there is no way for you to update that row. For rows that you are able to read, you must also be able to write to the row in order to update it.

When you try to update a row, your LBAC credentials for writing are compared to the security label protecting the row. If write access is blocked, the update fails and an error is returned. If write access is not blocked, then the update continues.

The update that is performed is done the same way as an update to a non-protected row except for the treatment of the column that has a data type of DB2SECURITYLABEL. If you do not explicitly set the value of that column, it is automatically set to the security label that you hold for write access. If you do not have a security label for write access, an error is returned and the statement fails.

If the update explicitly sets the column that has a data type of DB2SECURITYLABEL, then your LBAC credentials are checked again. If the update you are trying to perform would create a row that your current LBAC credentials would not allow you to write to, then what happens depends on the security policy that is protecting the table. If the security policy has the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option, then the update fails and an error is returned. If the security policy does not have the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option or if it instead has the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option, then the security label you provide is ignored and if you hold a security label for write access, it is used instead. If you do not hold a security label for write access, an error is returned.

Example:

Assume that table T1 is protected by a security policy named P1 and has a column named LABEL that has a data type of DB2SECURITYLABEL.

T1, including its data, looks like this:

Table 28. Table T1

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	11	L1
2	Miller	11	L2
3	Bird	11	L3

Assume that user Jenni has LBAC credentials that allow her to read and write data protected by the security labels L0 and L1 but not data protected by any other security labels. The security label she holds for both read and write is L0. The details of her full credentials and of what elements are in the labels are not important for this example.

Jenni issues this SQL statement:

```
SELECT * FROM T1
```

Jenni sees only one row in the table:

Table 29. Jenni's SELECT Query Result

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	11	L1

The rows protected by labels L2 and L3 are not included in the result set because Jenni's LBAC credentials do not allow her to read those rows. For Jenni it is as if those rows do not exist.

Jenni issues these SQL statements:

```
UPDATE T1 SET DEPTNO = 44 WHERE DEPTNO = 11;
SELECT * FROM T1;
```

The result set returned by the query looks like this:

Table 30. Jenni's UPDATE & SELECT Query Result

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L0

The actual data in the table looks like this:

Table 31. Table T1

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L0
2	Miller	11	L2
3	Bird	11	L3

The statement executed without error but affected only the first row. The second and third rows are not readable by Jenni so they are not selected for update by the statement even though they meet the condition in the WHERE clause.

Notice that the value of the LABEL column in the updated row has changed even though that column was not explicitly set in the UPDATE statement. The column was set to the security label that Jenni held for writing.

Now Jenni is granted LBAC credentials that allow her to read data protected by any security label. Her LBAC credentials for writing do not change. She is still only able to write to data protected by L0 and L1.

Jenni again issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 44 WHERE DEPTNO = 11
```

This time the update fails because of the second and third rows. Jenni is able to read those rows, so they are selected for update by the statement. She is not, however, able to write to them because they are protected by security labels L2 and L3. The update does not occur and an error is returned.

Jenni now issues this SQL statement:

```
UPDATE T1
 SET DEPTNO = 55, LABEL = SECLABEL_BY_NAME( 'P1', 'L2' )
 WHERE LASTNAME = "Rjaibi"
```

The SECLABEL_BY_NAME function in the statement returns the security label named L2. Jenni is trying to explicitly set the security label protecting the first row. Jenni's LBAC credentials allow her to read the first row, so it is selected for update. Her LBAC credentials allow her to write to rows protected by the security label L0 so she is allowed to update the row. Her LBAC credentials would not, however, allow her to write to a row protected by the security label L2, so she is not allowed to set the column LABEL to that value. The statement fails and an error is returned. No columns in the row are updated.

Jenni now issues this SQL statement:

```
UPDATE T1 SET LABEL = SECLABEL_BY_NAME( 'P1', 'L1' ) WHERE LASTNAME = "Rjaibi"
```

The statement succeeds because she would be able to write to a row protected by the security label L1.

T1 now looks like this:

Table 32. Table T1

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L1
2	Miller	11	L2
3	Bird	11	L3

Updating protected rows that contain protected columns

If you try to update protected columns in a table with protected rows then your LBAC credentials must allow writing to of all of the protected columns affected by the update, otherwise the update fails and an error is returned. This is as described in section about updating protected columns, earlier. If you are allowed to update all of the protected columns affected by the update you will still only be able to update rows that your LBAC credentials allow you to both read from and write to. This is as described in the section about updating protected rows, earlier. The handling of a column with a data type of DB2SECURITYLABEL is the same whether the update affects protected columns or not.

If the column that has a data type of DB2SECURITYLABEL is itself a protected column then your LBAC credentials must allow you to write to that column or you cannot update any of the rows in the table.

Deleting or dropping of LBAC protected data

Your ability to delete data in tables protected by LBAC depend on your LBAC credentials.

Deleting protected rows

If your LBAC credentials do not allow you to read a row, it is as if that row does not exist for you so there is no way for you to delete it. To delete a row that you are able to read, your LBAC credentials must also allow you to write to the row. To delete any row in a table that has protected columns you must have LBAC credentials that allow you to write to all protected columns in the table.

When you try to delete a row, your LBAC credentials for writing are compared to the security label protecting the row. If the protecting security label blocks write access by your LBAC credentials, the DELETE statement fails, an error is returned, and no rows are deleted.

Example

Protected table T1 has these rows:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2
Fielding	77	L3

Assume that user Pat has LBAC credentials such that her access is as summarized in this table:

Security label	Read access?	Write access?
L1	Yes	Yes
L2	Yes	No
L3	No	No

The exact details of her LBAC credentials and of the security labels are unimportant for this example.

Pat issues the following SQL statement:

```
SELECT * FROM T1 WHERE DEPTNO != 999
```

The statement executes and returns this result set:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2

The last row of T1 is not included in the results because Pat does not have read access to that row. It is as if that row does not exist for Pat.

Pat issues this SQL statement:

```
DELETE FROM T1 WHERE DEPTNO != 999
```

Pat does not have write access to the first or third row, both of which are protected by L2. So even though she can read the rows she cannot delete them. The DELETE statement fails and no rows are deleted.

Pat issues this SQL statement:

```
DELETE FROM T1 WHERE DEPTNO = 77;
```

This statement succeeds because Pat is able to write to the row with Miller in the LASTNAME column. That is the only row selected by the statement. The row with Fielding in the LASTNAME column is not selected because Pat's LBAC credentials do not allow her to read that row. That row is never considered for the delete so no error occurs.

The actual rows of the table now look like this:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Bird	55	L2
Fielding	77	L3

Deleting rows that have protected columns

To delete any row in a table that has protected columns you must have LBAC credentials that allow you to write to all protected columns in the table. If there is any row in the table that your LBAC credentials do not allow you to write to then the delete will fail and an error will be returned.

If the table has both protected columns and protected rows then to delete a particular row you must have LBAC credentials that allow you to write to every protected column in the table and also to read from and write to the row that you want to delete.

Example

In protected table T1, the column DEPTNO is protected by the security label L2. T1 contains these rows:

LASTNAME	**DEPTNO** *Protected by* **L2**	**LABEL**
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2
Fielding	77	L3

Assume that user Benny has LBAC credentials that allow him the access summarized in this table:

Security label	**Read access?**	**Write access?**
L1	Yes	Yes
L2	Yes	No
L3	No	No

The exact details of his LBAC credentials and of the security labels are unimportant for this example.

Benny issues the following SQL statement:

```
DELETE FROM T1 WHERE DEPTNO = 77
```

The statement fails because Benny does not have write access to the column DEPTNO.

Now Benny's LBAC credentials are changed so that he has access as summarized in this table:

Security label	**Read access?**	**Write access?**
L1	Yes	Yes
L2	Yes	Yes
L3	Yes	No

Benny issues this SQL statement again:

```
DELETE FROM T1 WHERE DEPTNO = 77
```

This time Benny has write access to the column DEPTNO so the delete continues. The delete statement selects only the row that has a value of Miller in the LASTNAME column. The row that has a value of Fielding in the LASTNAME column is not selected because Benny's LBAC credentials do not allow him to read that row. Because the row is not selected for deletion by the statement it does not matter that Benny is unable to write to the row.

The one row selected is protected by the security label L1. Benny's LBAC credentials allow him to write to data protected by L1 so the delete is successful.

The actual rows in table T1 now look like this:

LASTNAME	**DEPTNO** *Protected by* **L2**	**LABEL**
Rjaibi	55	L2
Bird	55	L2
Fielding	77	L3

Dropping protected data

You cannot drop a column that is protected by a security label unless your LBAC credentials allow you to write to that column.

A column with a data type of DB2SECURITYLABEL cannot be dropped from a table. To remove it you must first drop the security policy from the table. When you drop the security policy the table is no longer protected with LBAC and the data type of the column is automatically changed from DB2SECURITYLABEL to VARCHAR(128) FOR BIT DATA. The column can then be dropped.

Your LBAC credentials do not prevent you from dropping entire tables or databases that contain protected data. If you would normally have permission to drop a table or a database you do not need any LBAC credentials to do so, even if the database contains protected data.

Removal of LBAC protection from data

You must have SECADM authority to remove the security policy from a table. To remove the security policy from a table you use the DROP SECURITY POLICY clause of the ALTER TABLE statement. This also automatically removes protection from all rows and all columns of the table.

Removing protection from rows

In a table that has protected rows every row must be protected by a security label. There is no way to remove LBAC protection from individual rows.

A column of type DB2SECURITYLABEL cannot be altered or removed except by removing the security policy from the table.

Removing protection from columns

Protection of a column can be removed using the DROP COLUMN SECURITY clause of the SQL statement ALTER TABLE. To remove the protection from a column you must have LBAC credentials that allow you to read from and write to that column in addition to the normal privileges and authorities needed to alter a table.

LBAC-protected data load considerations

For a successful load operation into a table with protected rows, you must have LBAC (label-based access control) credentials. You must also provide a valid security label, or a security label that can be converted to a valid label, for the security policy currently associated with the target table.

If you do not have valid LBAC credentials, the load fails and an error (SQLSTATE 42512) is returned. In cases where the input data does not contain a security label or that security label is not in its internal binary format, you can use several file type modifiers to allow your load to proceed.

When you load data into a table with protected rows, the target table has one column with a data type of DB2SECURITYLABEL. If the input row of data does not contain a value for that column, that row is rejected unless the `usedefaults` file type modifier is specified in the load command, in which case the security label you hold for write access from the security policy protecting the table is used. If you do not hold a security label for write access, the row is rejected and processing continues on to the next row.

When you load data into a table that has protected rows and the input data does include a value for the column with a data type of DB2SECURITYLABEL, the same rules are followed as when you insert data into that table. If the security label protecting the row being loaded (the one in that row of the data file) is one that you are able to write to, then that security label is used to protect the row. (In other words, it is written to the column that has a data type of DB2SECURITYLABEL.) If you are not able to write to a row protected by that security label, what happens depends on how the security policy protecting the source table was created:

- If the CREATE SECURITY POLICY statement that created the policy included the option RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL, the row is rejected.
- If the CREATE SECURITY POLICY statement did not include the option or if it instead included the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option, the security label in the data file for that row is ignored and the security label you hold for write access is used to protect that row. No error or warning is issued in this case. If you do not hold a security label for write access, the row is rejected and processing continues on to the next row.

Delimiter considerations

When loading data into a column with a data type of DB2SECURITYLABEL, the value in the data file is assumed by default to be the actual bytes that make up the internal representation of that security label. However, some raw data might contain newline characters which could be misinterpreted by the LOAD command as delimiting the row. If you have this problem, use the `delprioritychar` file type modifier to ensure that the character delimiter takes precedence over the row delimiter. When you use `delprioritychar`, any record or column delimiters that

are contained within character delimiters are not recognized as being delimiters. Using the `delprioritychar` file type modifier is safe to do even if none of the values contain a newline character, but it does slow the load down slightly.

If the data being loaded is in ASC format, you might have to take an extra step in order to prevent any trailing white space from being included in the loaded security labels and security label names. ASCII format uses column positions as delimiters, so this might occur when loading into variable-length fields. Use the `striptblanks` file type modifier to truncate any trailing blank spaces.

Nonstandard security label values
You can also load data files in which the values for the security labels are strings containing the values of the components in the security label, for example, *S:(ALPHA,BETA)*. To do so you must use the file type modifier `seclabelchar`. When you use `seclabelchar`, a value for a column with a data type of DB2SECURITYLABEL is assumed to be a string constant containing the security label in the string format for security labels. If a string is not in the proper format, the row is not inserted and a warning (SQLSTATE 01H53) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table, the row is not inserted and a warning (SQLSTATE 01H53) is returned.

You can also load a data file in which the values of the security label column are security label names. To load this sort of file you must use the file type modifier `seclabelname`. When you use `seclabelname`, all values for columns with a data type of DB2SECURITYLABEL are assumed to be string constants containing the names of existing security labels. If no security label exists with the indicated name for the security policy protecting the table, the row is not loaded and a warning (SQLSTATE 01H53) is returned.

Rejected rows
Rows that are rejected during the load are sent to either a dumpfile or an exception table (if they are specified in the LOAD command), depending on the reason why the rows were rejected. Rows that are rejected due to parsing errors are sent to the dumpfile. Rows that violate security policies are sent to the exception table.

Note: You cannot specify an exception table if the target table contains an XML column.

Examples

For all examples, the input data file `myfile.del` is in DEL format. All are loading data into a table named REPS, which was created with this statement:

```
create table reps (row_label db2securitylabel,
id integer,
name char(30))
security policy data_access_policy
```

For this example, the input file is assumed to contain security labels in the default format:

```
db2 load from myfile.del of del modified by delprioritychar insert into reps
```

For this example, the input file is assumed to contain security labels in the security label string format:

```
db2 load from myfile.del of del modified by seclabelchar insert into reps
```

For this example, the input file is assumed to contain security labels names for the security label column:

```
db2 load from myfile.del of del modified by seclabelname insert into reps
```

Chapter 18. Gaining access to data through indirect means

To successfully manage security, you need to be aware of indirect ways that users can gain access to data.

The following are indirect means through which users can gain access to data they might not be authorized to access:

- **Catalog views:** The DB2 database system catalog views store metadata and statistics about database objects. Users with SELECT access to the catalog views can gain some knowledge about data that they might not be qualified for. For better security, make sure that only qualified users have access to the catalog views.

  Note: In DB2® Universal Database™ Version 8, or earlier, SELECT access on the catalog views was granted to PUBLIC by default. In DB2 Version 9.1, or later, database systems, users can choose whether SELECT access to the catalog views is granted to PUBLIC or not by using the new **RESTRICTIVE** option on the CREATE DATABASE command.
- **Visual explain:** Visual explain shows the access plan chosen by the query optimizer for a particular query. The visual explain information also includes statistics about columns referenced in the query. These statistics can reveal information about a table's contents.
- **Explain snapshot:** The explain snapshot is compressed information that is collected when an SQL or XQuery statement is explained. It is stored as a binary large object (BLOB) in the EXPLAIN_STATEMENT table, and contains column statistics that can reveal information about table data. For better security, access to the explain tables should be granted to qualified users only.
- **Log reader functions:** A user authorized to run a function that reads the logs can gain access to data they might not be authorized for if they are able to understand the format of a log record. These functions read the logs:

Function	Authority needed in order to execute the function
db2ReadLog	SYSADM or DBADM
db2ReadLogNoConn	None.

- **Replication:** When you replicate data, even the protected data is reproduced at the target location. For better security, make sure that the target location is at least as secure as the source location.
- **Exception tables:** When you specify an exception table while loading data into a table, users with access to the exception table can gain information that they might not be authorized for. For better security, only grant access to the exception table to authorized users and drop the exception table as soon as you are done with it.
- **Backup table space or database:** Users with the authority to run the BACKUP DATABASE command can take a backup of a database or a table space, including any protected data, and restore the data somewhere else. The backup can include data that the user might not otherwise have access to.

  The BACKUP DATABASE command can be executed by users with SYSADM, SYSCTRL, or SYSMAINT authority.
- **Set session authorization:** In DB2 Universal Database Version 8, or earlier, a user with DBADM authority could use the SET SESSION AUTHORIZATION

SQL statement to set the session authorization ID to any database user. In DB2 Version 9.1, or later, database systems a user must be explicitly authorized through the GRANT SETSESSIONUSER statement before they can set the session authorization ID.

When upgrading an existing Version 8 database to a DB2 Version 9.1, or later, database system, however, a user with existing explicit DBADM authority (for example, granted in SYSCAT.DBAUTH) will keep the ability to set the session authorization to any database user. This is allowed so that existing applications will continue to work. Being able to set the session authorization potentially allows access to all protected data. For more restrictive security, you can override this setting by executing the REVOKE SETSESSIONUSER SQL statement.

- **Statement and deadlock monitoring:** As part of the deadlock monitoring activity of DB2 database management systems, values associated with parameter markers are written to the monitoring output when the WITH VALUES clause is specified. A user with access to the monitoring output can gain access to information for which they might not be authorized.
- **Traces:** A trace can contain table data. A user with access to such a trace can gain access to information that they might not be authorized for.
- **Dump files:** To help in debugging certain problems, DB2 database products might generate memory dump files in the `sqllib\db2dump` directory. These memory dump files might contain table data. If they do, users with access to the files can gain access to information that they might not be authorized for. For better security you should limit access to the `sqllib\db2dump` directory.
- **db2dart:** The db2dart tool examines a database and reports any architectural errors that it finds. The tool can access table data and DB2 does not enforce access control for that access. A user with the authority to run the db2dart tool or with access to the db2dart output can gain access to information that they might not be authorized for.
- **REOPT bind option:** When the **REOPT** bind option is specified, explain snapshot information for each reoptimizable incremental bind SQL statement is placed in the explain tables at run time. The explain will also show input data values.
- **db2cat:** The db2cat tool is used to dump a table's packed descriptor. The table's packed descriptor contains statistics that can reveal information about a table's contents. A user who runs the db2cat tool or has access to the output can gain access to information that they might not be authorized for.

Chapter 19. Authorization ID privileges: SETSESSIONUSER

Authorization ID privileges involve actions on authorization IDs. There is currently only one such privilege: the SETSESSIONUSER privilege.

The SETSESSIONUSER privilege can be granted to a user or to a group and allows the holder to switch identities to any of the authorization IDs on which the privilege was granted. The identity switch is made by using the SQL statement SET SESSION AUTHORIZATION. The SETSESSIONUSER privilege can only be granted by a user holding SECADM authority.

Note: When you upgrade a Version 8 database to Version 9.1, or later, authorization IDs with explicit DBADM authority on that database are automatically granted SETSESSIONUSER privilege on PUBLIC. This prevents breaking applications that rely on authorization IDs with DBADM authority being able to set the session authorization ID to any authorization ID. This does not happen when the authorization ID has SYSADM authority but has not been explicitly granted DBADM.

Chapter 20. User responsibilities for security

In the DB2 database manager, access to the database and its resources are typically controlled by the system administrator (SYSADM) or database administrator (DBADM) through the use facilities such as the GRANT and REVOKE statements.

Database users also have an important role in maintaining the security of the database, and the objects and data in the database. At a minimum, you should implement the following guidelines to protect against unauthorized access to your workstation, the database, and data within the database:

- Lock your workstation when you are away from your desk.
- Ensure that you use a user ID and a password to log on to your computer and to log on to the database manager.

  If you are working on an operating system that allows for NULL passwords (that is, you do not have to supply a password to log on to your machine), explicitly define a password for your user ID.

  Passwords are ordinarily a minimum of 8 characters, and should be changed regularly. If you are not certain about the practices followed at your site, check your site's security regulations.
- Do not give your password to unauthorized users.

  Some sites require that you provide your manager with your password. If you are not certain about the practices followed at your site, check your site's security regulations.
- If an unexpected event occurs, or you suspect that an unauthorized individual has accessed either your workstation or the database manager, contact your manager or database administrator immediately.
- When using the CONNECT statement to connect to the database from the command line processor (CLP), allow the database manager to prompt you for the password, rather than entering it with the CONNECT statement. This practice prevents the password from being entered into the command history of the operating system.
- Do not hardcode passwords into applications that connect to the database.

Chapter 21. Extended Windows security using the DB2ADMNS and DB2USERS groups

Extended security is enabled by default in all DB2 database products on Windows operating systems except IBM Data Server Runtime Client and DB2 Drivers. IBM Data Server Runtime Client and DB2 Drivers do not support extended security on Windows platforms.

An **Enable operating system security** check box appears on the **Enable operating system security for DB2 objects** panel when you install DB2 database products. Unless you disable this option, the installer creates two new groups, DB2ADMNS and DB2USERS. DB2ADMNS and DB2USERS are the default group names; optionally, you can specify different names for these groups at installation time (if you select silent install, you can change these names within the install response file). If you choose to use groups that already exist on your system, be aware that the privileges of these groups will be modified. They will be given the privileges, as required, listed in the table, below. It is important to understand that these groups are used for protection at the *operating-system* level and are in no way associated with DB2 authority levels, such as SYSADM, SYSMAINT, and SYSCTRL. However, instead of using the default Administrator's group, your database administrator can use the DB2ADMNS group for one or all of the DB2 authority levels, at the discretion of the installer or administrator. It is recommended that if you are specifying a SYSADM group, then that should be the DB2ADMNS group. This can be established during installation or subsequently, by an administrator.

Note: You can specify your DB2 Administrators Group (DB2ADMNS or the name you chose during installation) and DB2 Users Group (DB2USERS or the name you chose during installation) either as local groups or as domain groups. Both groups must be of the same type, so either both local or both domain.

If you change the computer name, and the computer groups DB2ADMNS and DB2USERS are local computer groups, you must update the DB2_ADMINGROUP and DB2_USERSGROUP global registries. To update the registry variables after renaming and restarting the computer run the following command:

1. Open a command prompt.
2. Run the db2extsec command to update security settings:

   ```
   db2extsec -a new computer name\DB2ADMNS -u new computer name\DB2USERS
   ```

Note: If extended security is enabled in DB2 database products on Windows Vista, only users that belong to the DB2ADMNS group can run the graphical DB2 administration tools. In addition, members of the DB2ADMNS group need to launch the tools with full administrator privileges. This is accomplished by right-clicking on the shortcut and then choosing "Run as administrator".

Abilities acquired through the DB2ADMNS and DB2USERS groups

The DB2ADMNS and DB2USERS groups provide members with the following abilities:

- DB2ADMNS

  Full control over all DB2 objects (see the list of protected objects, below)

- DB2USERS

  Read and Execute access for all DB2 objects located in the installation and instance directories, but no access to objects under the database system directory and limited access to IPC resources

  For certain objects, there may be additional privileges available, as required (for example, write privileges, add or update file privileges, and so on). Members of this group have no access to objects under the database system directory.

  Note: The meaning of Execute access depends on the object; for example, for a **.dll** or **.exe** file having Execute access means you have authority to execute the file, however, for a directory it means you have authority to traverse the directory.

Ideally, all DB2 administrators should be members of the DB2ADMNS group (as well as being members of the local Administrators group), but this is not a strict requirement. Everyone else who requires access to the DB2 database system *must* be a member of the DB2USERS group. To add a user to one of these groups:

1. Launch the Users and Passwords Manager tool.
2. Select the user name to add from the list.
3. Click Properties. In the Properties window, click the Group membership tab.
4. Select the Other radio button.
5. Select the appropriate group from the drop-down list.

Adding extended security after installation (db2extsec command)

If the DB2 database system was installed without extended security enabled, you can enable it by executing the command **db2extsec**. To execute the **db2extsec** command you must be a member of the local Administrators group so that you have the authority to modify the ACL of the protected objects.

You can run the **db2extsec** command multiple times, if necessary, however, if this is done, you cannot disable extended security unless you issue the **db2extsec –r** command immediately after *each* execution of **db2extsec**.

Removing extended security

CAUTION:
Do not remove extended security after it has been enabled unless absolutely necessary.

You can remove extended security by running the command **db2extsec -r**, however, this will only succeed if no other database operations (such as creating a database, creating a new instance, adding table spaces, and so on) have been performed after enabling extended security. The safest way to remove the extended security option is to uninstall the DB2 database system, delete all the relevant DB2 directories (including the database directories) and then reinstall the DB2 database system without extended security enabled.

Protected objects

The *static* objects that can be protected using the DB2ADMNS and DB2USERS groups are:

- File system
  - File

  - Directory
- Services
- Registry keys

The *dynamic* objects that can be protected using the DB2ADMNS and DB2USERS groups are:

- IPC resources, including:
  - Pipes
  - Semaphores
  - Events
- Shared memory

Privileges owned by the DB2ADMNS and DB2USERS groups

The privileges assigned to the DB2ADMNS and DB2USERS groups are listed in the following table:

Table 33. Privileges for DB2ADMNS and DB2USERS groups

Privilege	DB2ADMNS	DB2USERS	Reason
Create a token object (SeCreateTokenPrivilege)	Y	N	Token manipulation (required for certain token manipulation operations and used in authentication and authorization)
Replace a process level token (SeAssignPrimaryTokenPrivilege)	Y	N	Create process as another user
Increase quotas (SeIncreaseQuotaPrivilege)	Y	N	Create process as another user
Act as part of the operating system (SeTcbPrivilege)	Y	N	LogonUser (required prior to Windows XP in order to execute the LogonUser API for authentication purposes)
Generate security audits (SeSecurityPrivilege)	Y	N	Manipulate audit and security log
Take ownership of files or other objects (SeTakeOwnershipPrivilege)	Y	N	Modify object ACLs
Increase scheduling priority (SeIncreaseBasePriorityPrivilege)	Y	N	Modify the process working set
Backup files and directories (SeBackupPrivilege)	Y	N	Profile/Registry manipulation (required to perform certain user profile and registry manipulation routines: LoadUserProfile, RegSaveKey(Ex), RegRestoreKey, RegReplaceKey, RegLoadKey(Ex))
Restore files and directories (SeRestorePrivilege)	Y	N	Profile/Registry manipulation (required to perform certain user profile and registry manipulation routines: LoadUserProfile, RegSaveKey(Ex), RegRestoreKey, RegReplaceKey, RegLoadKey(Ex))
Debug programs (SeDebugPrivilege)	Y	N	Token manipulation (required for certain token manipulation operations and used in authentication and authorization)
Manage auditing and security log (SeAuditPrivilege)	Y	N	Generate auditing log entries
Log on as a service (SeServiceLogonRight)	Y	N	Run DB2 as a service

Table 33. Privileges for DB2ADMNS and DB2USERS groups (continued)

Privilege	DB2ADMNS	DB2USERS	Reason
Access this computer from the network (SeNetworkLogonRight)	Y	Y	Allow network credentials (allows the DB2 database manager to use the LOGON32_LOGON_NETWORK option to authenticate, which has performance implications)
Impersonate a client after authentication (SeImpersonatePrivilege)	Y	N	Client impersonation (required for Windowsto allow use of certain APIs to impersonate DB2 clients: ImpersonateLoggedOnUser, ImpersonateSelf, RevertToSelf, and so on)
Lock pages in memory (SeLockMemoryPrivilege)	Y	N	Large Page support
Create global objects (SeCreateGlobalPrivilege)	Y	Y	Terminal Server support (required on Windows)

Chapter 22. Roles

Roles

Roles simplify the administration and management of privileges by offering an equivalent capability as groups but without the same restrictions.

A role is a database object that groups together one or more privileges and can be assigned to users, groups, PUBLIC, or other roles by using a GRANT statement, or can be assigned to a trusted context by using a CREATE TRUSTED CONTEXT or ALTER TRUSTED CONTEXT statement. A role can be specified for the SESSION_USER ROLE connection attribute in a workload definition.

Roles provide several advantages that make it easier to manage privileges in a database system:

- Security administrators can control access to their databases in a way that mirrors the structure of their organizations (they can create roles in the database that map directly to the job functions in their organizations).
- Users are granted membership in the roles that reflect their job responsibilities. As their job responsibilities change, their membership in roles can be easily granted and revoked.
- The assignment of privileges is simplified. Instead of granting the same set of privileges to each individual user in a particular job function, the administrator can grant this set of privileges to a role representing that job function and then grant that role to each user in that job function.
- A role's privileges can be updated and all users who have been granted that role receive the update; the administrator does not need to update the privileges for every user on an individual basis.
- The privileges and authorities granted to roles are always used when you create views, triggers, materialized query tables (MQTs), static SQL and SQL routines, whereas privileges and authorities granted to groups (directly or indirectly) are not used.

  This is because the DB2 database system cannot determine when membership in a group changes, as the group is managed by third-party software (for example, the operating system or an LDAP directory). Because roles are managed inside the database, the DB2 database system can determine when authorization changes and act accordingly. Roles granted to groups are not considered, due to the same reason groups are not considered.
- All the roles assigned to a user are enabled when that user establishes a connection, so all privileges and authorities granted to roles are taken into account when a user connects. Roles cannot be explicitly enabled or disabled.
- The security administrator can delegate management of a role to others.

All DB2 privileges and authorities that can be granted within a database can be granted to a role. For example, a role can be granted any of the following authorities and privileges:

- DBADM, SECADM, DATAACCESS, ACCESSCTRL, SQLADM, WLMADM, LOAD, and IMPLICIT_SCHEMA database authorities
- CONNECT, CREATETAB, CREATE_NOT_FENCED, BINDADD, CREATE_EXTERNAL_ROUTINE, or QUIESCE_CONNECT database authorities

- Any database object privilege (including CONTROL)

A user's roles are automatically enabled and considered for authorization when a user connects to a database; you do not need to activate a role by using the SET ROLE statement. For example, when you create a view, a materialized query table (MQT), a trigger, a package, or an SQL routine, the privileges that you gain through roles apply. However, privileges that you gain through roles granted to groups of which you are a member do not apply.

A role does not have an owner. The security administrator can use the WITH ADMIN OPTION clause of the GRANT statement to delegate management of the role to another user, so that the other user can control the role membership.

Restrictions

There are a few restrictions in the use of roles:

- A role cannot own database objects.
- Permissions and roles granted to groups are not considered when you create the following database objects:
  - Packages containing static SQL
  - Views
  - Materialized query tables (MQT)
  - Triggers
  - SQL Routines

  Only roles granted to the user creating the object or to PUBLIC, directly or indirectly (such as through a role hierarchy), are considered when creating these objects.

Creating and granting membership in roles

The security administrator holds the authority to create, drop, grant, revoke, and comment on a role. The security administrator uses the GRANT (Role) statement to grant membership in a role to an authorization ID and uses the REVOKE (Role) statement to revoke membership in a role from an authorization ID.

The security administrator can delegate the management of membership in a role to an authorization ID by granting the authorization ID membership in the role with the WITH ADMIN OPTION. The WITH ADMIN OPTION clause of the GRANT (Role) statement gives another user the ability to:

- Grant roles to others.
- Revoke roles from others.
- Comment on the role.

The WITH ADMIN OPTION clause does not give the ability to:

- Drop the role.
- Revoke the WITH ADMIN OPTION for a role from an authorization ID.
- Grant WITH ADMIN OPTION to someone else (if you do not hold SECADM authority).

After the security administrator has created a role, the database administrator can use the GRANT statement to assign authorities and privileges to the role. All DB2

privileges and authorities that can be granted within a database can be granted to a role. Instance level authorities, such as SYSADM authority, cannot be assigned to a role.

The security administrator, or any user who the security administrator has granted membership in a role with WITH ADMIN OPTION can use the GRANT (Role) statement to grant membership in that role to other users, groups, PUBLIC or roles. A user may have been granted membership in a role with WITH ADMIN OPTION either directly, or indirectly through PUBLIC, a group or a role.

All the roles assigned to a user are enabled when that user establishes a session. All the privileges and authorities associated with a user's roles are taken into account when the DB2 database system checks for authorization. Some database systems use the SET ROLE statement to activate a particular role. The DB2 database system supports SET ROLE to provide compatibility with other products using the SET ROLE statement. In a DB2 database system, the SET ROLE statement checks whether the session user is a member of the role and returns an error if they are not.

To revoke a user's membership in a role, the security administrator, or a user who holds WITH ADMIN OPTION privilege on the role, uses the REVOKE (Role) statement.

Example

A role has a certain set of privileges and a user who is granted membership in this role inherits those privileges. This inheritance of privileges eliminates managing individual privileges when reassigning the privileges of one user to another user. The only operations required when using roles is to revoke membership in the role from one user and grant membership in the role to the other user.

For example, the employees BOB and ALICE, working in department DEV, have the privilege to SELECT on the tables SERVER, CLIENT and TOOLS. One day, management decides to move them to a new department, QA, and the database administrator has to revoke their privilege to select on tables SERVER, CLIENT and TOOLS. Department DEV later hires a new employee, TOM, and the database administrator has to grant SELECT privilege on tables SERVER, CLIENT and TOOLS to TOM.

When using roles, the following steps occur:

1. The security administrator creates a role, DEVELOPER:
   ```
   CREATE ROLE DEVELOPER
   ```
2. The database administrator (who holds DBADM authority) grants SELECT on tables SERVER, CLIENT, and TOOLS to role DEVELOPER:
   ```
   GRANT  SELECT ON TABLE SERVER TO ROLE DEVELOPER
   GRANT  SELECT ON TABLE CLIENT TO ROLE DEVELOPER
   GRANT  SELECT ON TABLE TOOLS TO ROLE DEVELOPER
   ```
3. The security administrator grants the role DEVELOPER to the users in department DEV, BOB and ALICE:
   ```
   GRANT ROLE DEVELOPER TO USER BOB, USER ALICE
   ```
4. When BOB and ALICE leave department DEV, the security administrator revokes the role DEVELOPER from users BOB and ALICE:
   ```
   REVOKE ROLE DEVELOPER FROM USER BOB, USER ALICE
   ```

5. When TOM is hired in department DEV, the security administrator grants the role DEVELOPER to user TOM:

```
GRANT ROLE DEVELOPER TO USER TOM
```

Role hierarchies

A role hierarchy is formed when one role is granted membership in another role.

A role *contains* another role when the other role is granted to the first role. The other role inherits all of the privileges of the first role. For example, if the role DOCTOR is granted to the role SURGEON, then SURGEON is said to contain DOCTOR. The role SURGEON inherits all the privileges of role DOCTOR.

Cycles in role hierarchies are not allowed. A *cycle* occurs if a role is granted in circular way such that one role is granted to another role and that other role is granted to the original role. For example, the role DOCTOR is granted to role SURGEON, and then the role SURGEON is granted back to the role DOCTOR. If you create a cycle in a role hierarchy, an error is returned (SQLSTATE 428GF).

Example of building a role hierarchy

The following example shows how to build a role hierarchy to represent the medical levels in a hospital.

Consider the following roles: DOCTOR, SPECIALIST, and SURGEON. A role hierarchy is built by granting a role to another role, but without creating cycles. The role DOCTOR is granted to role SPECIALIST, and role SPECIALIST is granted to role SURGEON.

Granting role SURGEON to role DOCTOR would create a cycle and is not allowed.

The security administrator runs the following SQL statements to build the role hierarchy:

```
CREATE ROLE DOCTOR
CREATE ROLE SPECIALIST
CREATE ROLE SURGEON

GRANT ROLE DOCTOR TO ROLE SPECIALIST

GRANT ROLE SPECIALIST TO ROLE SURGEON
```

Effect of revoking privileges from roles

When privileges are revoked, this can sometimes cause dependent database objects, such as views, packages or triggers, to become invalid or inoperative.

The following examples show what happens to a database object when some privileges are revoked from an authorization identifier and privileges are held through a role or through different means.

Example of revoking privileges from roles

1. The security administrator creates the role DEVELOPER and grants the user BOB membership in this role:

   ```
   CREATE ROLE DEVELOPER
   GRANT ROLE DEVELOPER TO USER BOB
   ```

2. User ALICE creates a table, WORKITEM:

   ```
   CREATE TABLE WORKITEM (x int)
   ```

3. The database administrator grants SELECT and INSERT privileges on table WORKITEM to PUBLIC and also to the role DEVELOPER:

```
GRANT SELECT ON TABLE ALICE.WORKITEM TO PUBLIC
GRANT INSERT ON TABLE ALICE.WORKITEM TO PUBLIC
GRANT SELECT ON TABLE ALICE.WORKITEM TO ROLE DEVELOPER
GRANT INSERT ON TABLE ALICE.WORKITEM TO ROLE DEVELOPER
```

4. User BOB creates a view, PROJECT, that uses the table WORKITEM, and a package, PKG1, that depends on the table WORKITEM:

```
CREATE VIEW PROJECT AS SELECT * FROM ALICE.WORKITEM
PREP emb001.sqc BINDFILE PACKAGE USING PKG1 VERSION 1
```

5. If the database administrator revokes SELECT privilege on table ALICE.WORKITEM from PUBLIC, then the view BOB.PROJECT remains operative and package PKG1 remains valid because the view definer, BOB, still holds the privileges required through his membership in the role DEVELOPER:

```
REVOKE SELECT ON TABLE ALICE.WORKITEM FROM PUBLIC
```

6. If the database administrator revokes SELECT privilege on table ALICE.WORKITEM from the role DEVELOPER, the view BOB.PROJECT becomes inoperative and package PKG1 becomes invalid because the view and package definer, BOB, does not hold the required privileges through other means:

```
REVOKE SELECT ON TABLE ALICE.WORKITEM FROM ROLE DEVELOPER
```

Example of revoking DBADM authority

In this example, the role DEVELOPER holds DBADM authority and is granted to user BOB.

1. The security administrator creates the role DEVELOPER:

```
CREATE ROLE DEVELOPER
```

2. The system administrator grants DBADM authority to the role DEVELOPER:

```
GRANT DBADM ON DATABASE TO ROLE DEVELOPER
```

3. The security administrator grants user BOB membership in this role:

```
GRANT ROLE DEVELOPER TO USER BOB
```

4. User ALICE creates a table, WORKITEM:

```
CREATE TABLE WORKITEM (x int)
```

5. User BOB creates a view PROJECT that uses table WORKITEM, a package PKG1 that depends on table WORKITEM, and a trigger, TRG1, that also depends on table WORKITEM:

```
CREATE VIEW PROJECT AS SELECT * FROM ALICE.WORKITEM
PREP emb001.sqc BINDFILE PACKAGE USING PKG1 VERSION 1
CREATE TRIGGER TRG1 AFTER DELETE ON ALICE.WORKITEM
              FOR EACH STATEMENT MODE DB2SQL
       INSERT INTO ALICE.WORKITEM VALUES (1)
```

6. The security administrator revokes the role DEVELOPER from user BOB:

```
REVOKE ROLE DEVELOPER FROM USER BOB
```

   Revoking the role DEVELOPER causes the user BOB to lose DBADM authority because the role that held that authority was revoked. The view, package, and trigger are affected as follows:

   - View BOB. PROJECT is still valid.
   - Package PKG1 becomes invalid.
   - Trigger BOB.TRG1 is still valid.

View BOB.PROJECT and trigger BOB.TRG1 are usable while package PKG1 is not usable. View and trigger objects created by an authorization ID holding DBADM authority are not affected when DBADM authority is lost.

Delegating role maintenance by using the WITH ADMIN OPTION clause

Using the WITH ADMIN OPTION clause of the GRANT (Role) SQL statement, the security administrator can delegate the management and control of membership in a role to someone else.

The WITH ADMIN OPTION clause gives another user the authority to grant membership in the role to other users, to revoke membership in the role from other members of the role, and to comment on a role, but not to drop the role.

The WITH ADMIN OPTION clause does not give another user the authority to grant WITH ADMIN OPTION on a role to another user. It also does not give the authority to revoke WITH ADMIN OPTION for a role from another authorization ID.

Example demonstrating use of the WITH ADMIN OPTION clause

1. A security administrator creates the role, DEVELOPER, and grants the new role to user BOB using the WITH ADMIN OPTION clause:

   ```
   CREATE ROLE DEVELOPER
   GRANT ROLE DEVELOPER TO USER BOB WITH ADMIN OPTION
   ```

2. User BOB can grant membership in the role to and revoke membership from the role from other users, for example, ALICE:

   ```
   GRANT ROLE DEVELOPER TO USER ALICE
   REVOKE ROLE DEVELOPER FROM USER ALICE
   ```

3. User BOB cannot drop the role or grant WITH ADMIN OPTION to another user (only a security administrator can perform these two operations). These commands issued by BOB will fail:

   ```
   DROP ROLE DEVELOPER  - FAILURE!
                        - only a security administrator is allowed to drop the role
   GRANT ROLE DEVELOPER TO USER ALICE WITH ADMIN OPTION - FAILURE!
                        - only a security administrator can grant WITH ADMIN OPTION
   ```

4. User BOB cannot revoke role administration privileges (conferred by WITH ADMIN OPTION) from users for role DEVELOPER, because he does not have security administrator (SECADM) authority. When BOB issues the following command, it fails:

   ```
   REVOKE ADMIN OPTION FOR ROLE DEVELOPER FROM USER SANJAY - FAILURE!
   ```

5. A security administrator is allowed to revoke the role administration privileges for role DEVELOPER (conferred by WITH ADMIN OPTION) from user BOB , and user BOB still has the role DEVELOPER granted:

   ```
   REVOKE ADMIN OPTION FOR ROLE DEVELOPER FROM USER BOB
   ```

   Alternatively, if a security administrator simply revokes the role DEVELOPER from user BOB, then BOB loses all the privileges he received by being a member of the role DEVELOPER and the authority on the role he received through the WITH ADMIN OPTION clause:

   ```
   REVOKE ROLE DEVELOPER FROM USER BOB
   ```

Roles compared to groups

Privileges and authorities granted to groups are not considered when creating views, materialized query tables (MQTs), SQL routines, triggers, and packages containing static SQL. Avoid this restriction by using roles instead of groups.

Roles allow users to create database objects using their privileges acquired through roles, which are controlled by the DB2 database system. Groups and users are controlled externally from the DB2 database system, for example, by an operating system or an LDAP server.

Example of replacing the use of groups with roles

This example shows how you can replace groups by using roles.

Assume there are three groups, DEVELOPER_G, TESTER_G and SALES_G. The users BOB, ALICE, and TOM are members of these groups, as shown in the following table:

Table 34. Example groups and users

Group	Users belonging to this group
DEVELOPER_G	BOB
TESTER_G	ALICE, TOM
SALES_G	ALICE, BOB

1. The security administrator creates the roles DEVELOPER, TESTER and SALES to be used instead of the groups.

   ```
   CREATE ROLE DEVELOPER
   CREATE ROLE TESTER
   CREATE ROLE SALES
   ```

2. The security administrator grants membership in these roles to users (setting the membership of users in groups was the system administrator's responsibility):

   ```
   GRANT  ROLE DEVELOPER TO USER BOB
   GRANT  ROLE TESTER TO USER ALICE, USER TOM
   GRANT  ROLE SALES TO USER BOB, USER ALICE
   ```

3. The database administrator can grant to the roles similar privileges or authorities as were held by the groups, for example:

   ```
   GRANT <privilege> ON <object> TO ROLE DEVELOPER
   ```

   The database administrator can then revoke these privileges from the groups, as well as ask the system administrator to remove the groups from the system.

Example of creating a trigger using privileges acquired through a role

This example shows that user BOB can successfully create a trigger, TRG1, when he holds the necessary privilege through the role DEVELOPER.

1. First, user ALICE creates the table, WORKITEM:

   ```
   CREATE TABLE WORKITEM (x int)
   ```

2. Then, the privilege to alter ALICE's table is granted to role DEVELOPER by the database administrator.

   ```
   GRANT ALTER ON ALICE.WORKITEM TO ROLE DEVELOPER
   ```

3. User BOB successfully creates the trigger, TRG1, because he is a member of the role, DEVELOPER.

```
CREATE TRIGGER TRG1 AFTER DELETE ON ALICE.WORKITEM
           FOR EACH STATEMENT MODE DB2SQL INSERT INTO ALICE.WORKITEM VALUES (1)
```

Using roles after migrating from IBM Informix Dynamic Server

If you have migrated from IBM Informix® Dynamic Server to the DB2 database system and are using roles there are a few things you need to be aware of.

The Informix Dynamic Server (IDS) SQL statement, GRANT ROLE, provides the clause WITH GRANT OPTION. The DB2 database system GRANT ROLE statement provides the clause WITH ADMIN OPTION (this conforms to the SQL standard) that provides the same functionality. During an IDS to DB2 database system migration, after the dbschema tool generates CREATE ROLE and GRANT ROLE statements, the dbschema tool replaces any occurrences of WITH GRANT OPTION with WITH ADMIN OPTION.

In an IDS database system, the SET ROLE statement activates a particular role. The DB2 database system supports the SET ROLE statement, but only to provide compatibility with other products using that SQL statement. The SET ROLE statement checks whether the session user is a member of the role and returns an error if they are not.

Example dbschema output

Assume that an IDS database contains the roles DEVELOPER, TESTER and SALES. Users BOB, ALICE, and TOM have different roles granted to each of them; the role DEVELOPER is granted to BOB, the role TESTER granted to ALICE, and the roles TESTER and SALES granted to TOM. To migrate to the DB2 database system, use the dbschema tool to generate the CREATE ROLE and GRANT ROLE statements for the database:

```
CREATE ROLE DEVELOPER
CREATE ROLE TESTER
CREATE ROLE SALES

GRANT  DEVELOPER TO BOB
GRANT  TESTER TO ALICE, TOM
GRANT  SALES TO TOM
```

You must create the database in the DB2 database system, and then you can run the above statements in that database to recreate the roles and assignment of the roles.

Chapter 23. Trusted contexts

Using trusted contexts and trusted connections

You can establish an explicit trusted connection by making a request within an application when a connection to a DB2 database is established. The security administrator must have previously defined a trusted context, using the CREATE TRUSTED CONTEXT statement, with attributes matching those of the connection you are establishing (see Step 1, later).

Before you begin

The API you use to request an explicit trusted connection when you establish a connection depends on the type of application you are using (see the table in Step 2).

After you have established an explicit trusted connection, the application can switch the user ID of the connection to a different user ID using the appropriate API for the type of application (see the table in Step 3).

1. The security administrator defines a trusted context in the server by using the CREATE TRUSTED CONTEXT statement. For example:

```
CREATE TRUSTED CONTEXT MYTCX
       BASED UPON CONNECTION USING SYSTEM AUTHID NEWTON
       ATTRIBUTES (ADDRESS   '192.0.2.1')
       WITH USE FOR PUBLIC WITHOUT AUTHENTICATION
       ENABLE
```

2. To establish a trusted connection, use one of the following APIs in your application:

Option	Description
Application	**API**
CLI/ODBC	SQLConnect, SQLSetConnectAttr
XA CLI/ODBC	Xa_open
JAVA	getDB2TrustedPooledConnection, getDB2TrustedXAConnection

3. To switch to a different user, with or without authentication, use one of the following APIs in your application:

Option	Description
Application	**API**
CLI/ODBC	SQLSetConnectAttr
XA CLI/ODBC	SQLSetConnectAttr
JAVA	getDB2Connection, reuseDB2Connection
.NET	DB2Connection.ConnectionString keywords: TrustedContextSystemUserID and TrustedContextSystemPassword

The switching can be done either with or without authenticating the new user ID, depending on the definition of the trusted context object associated with the

explicit trusted connection. For example, suppose that the security administrator creates the following trusted context object:

```
CREATE TRUSTED CONTEXT CTX1
 BASED UPON CONNECTION USING SYSTEM AUTHID USER1
 ATTRIBUTES (ADDRESS   '192.0.2.1')
  WITH USE FOR USER2 WITH AUTHENTICATION,
               USER3 WITHOUT AUTHENTICATION
  ENABLE
```

Further, suppose that an explicit trusted connection is established. A request to switch the user ID on the trusted connection to USER3 without providing authentication information is allowed because USER3 is defined as a user of trusted context CTX1 for whom authentication is not required. However, a request to switch the user ID on the trusted connection to USER2 without providing authentication information will fail because USER2 is defined as a user of trusted context CTX1 for whom authentication information must be provided.

Example of establishing an explicit trusted connection and switching the user

In the following example, a middle-tier server needs to issue some database requests on behalf of an end-user, but does not have access to the end-user's credentials to establish a database connection on behalf of that end-user.

You can create a trusted context object on the database server that allows the middle-tier server to establish an explicit trusted connection to the database. After establishing an explicit trusted connection, the middle-tier server can switch the current user ID of the connection to a new user ID without the need to authenticate the new user ID at the database server. The following CLI code snippet demonstrates how to establish a trusted connection using the trusted context, MYTCX, defined in Step 1, earlier, and how to switch the user on the trusted connection without authentication.

```
int main(int argc, char *argv[])
{
 SQLHANDLE henv;         /* environment handle */
 SQLHANDLE hdbc1;       /* connection handle */
 char origUserid[10] = "newton";
  char password[10] = "test";
   char switchUserid[10] = "zurbie";
 char dbName[10] = "testdb";

// Allocate the handles
SQLAllocHandle( SQL_HANDLE_ENV, &henv );
SQLAllocHandle( SQL_HANDLE_DBC, &hdbc1 );

// Set the trusted connection attribute
SQLSetConnectAttr( hdbc1, SQL_ATTR_USE_TRUSTED_CONTEXT,
SQL_TRUE, SQL_IS_INTEGER );

// Establish a trusted connection
SQLConnect( hdbc1, dbName, SQL_NTS, origUserid, SQL_NTS,
password, SQL_NTS );

//Perform some work under user ID "newton"
. . . . . .

 // Commit the work
SQLEndTran(SQL_HANDLE_DBC, hdbc1, SQL_COMMIT);
```

```
// Switch the user ID on the trusted connection
SQLSetConnectAttr( hdbc1,
SQL_ATTR_TRUSTED_CONTEXT_USERID, switchUserid,
SQL_IS_POINTER
);

//Perform new work using user ID "zurbie"
. . . . . .

//Commit the work
SQLEndTranSQL_HANDLE_DBC, hdbc1, SQL_COMMIT);

// Disconnect from database
SQLDisconnect( hdbc1 );

 return 0;

}  /* end of main */
```

What to do next

When does the user ID actually get switched?

After the command to switch the user on the trusted connection is issued, the switch user request is not performed until the next statement is sent to the server. This is demonstrated by the following example where the list applications command shows the original user ID until the next statement is issued.

1. Establish an explicit trusted connection with USERID1.
2. Issue the switch user command, such as getDB2Connection for USERID2.
3. Run `db2 list applications`. It still shows that USERID1 is connected.
4. Issue a statement on the trusted connection, such as `executeQuery("values current sqlid")`, to perform the switch user request at the server.
5. Run `db2 list applications` again. It now shows that USERID2 is connected.

Trusted contexts and trusted connections

A trusted context is a database object that defines a trust relationship for a connection between the database and an external entity such as an application server.

The trust relationship is based upon the following set of attributes:

- System authorization ID: Represents the user that establishes a database connection
- IP address (or domain name): Represents the host from which a database connection is established
- Data stream encryption: Represents the encryption setting (if any) for the data communication between the database server and the database client

When a user establishes a database connection, the DB2 database system checks whether the connection matches the definition of a trusted context object in the database. When a match occurs, the database connection is said to be trusted.

A trusted connection allows the initiator of this trusted connection to acquire additional capabilities that may not be available outside the scope of the trusted connection. The additional capabilities vary depending on whether the trusted connection is explicit or implicit.

The initiator of an explicit trusted connection has the ability to:

- Switch the current user ID on the connection to a different user ID with or without authentication
- Acquire additional privileges via the role inheritance feature of trusted contexts

An implicit trusted connection is a trusted connection that is not explicitly requested; the implicit trusted connection results from a normal connection request rather than an explicit trusted connection request. No application code changes are needed to obtain an implicit connection. Also, whether you obtain an implicit trusted connection or not has no effect on the connect return code (when you request an explicit trusted connection, the connect return code indicates whether the request succeeds or not). The initiator of an implicit trusted connection can only acquire additional privileges via the role inheritance feature of trusted contexts; they cannot switch the user ID.

How using trusted contexts enhances security

The three-tiered application model extends the standard two-tiered client and server model by placing a middle tier between the client application and the database server. It has gained great popularity in recent years particularly with the emergence of web-based technologies and the Java™ 2 Enterprise Edition (J2EE) platform. An example of a software product that supports the three-tier application model is IBM WebSphere® Application Server (WAS).

In a three-tiered application model, the middle tier is responsible for authenticating the users running the client applications and for managing the interactions with the database server. Traditionally, all the interactions with the database server occur through a database connection established by the middle tier using a combination of a user ID and a credential that identify that middle tier to the database server. In other words, the database server uses the database privileges associated with the middle tier's user ID for all authorization checking and auditing that must occur for any database access, including access performed by the middle tier on behalf of a user.

While the three-tiered application model has many benefits, having all interactions with the database server (for example, a user request) occur under the middle tier's authorization ID raises several security concerns, which can be summarized as follows:

- Loss of user identity

  Some enterprises prefer to know the identity of the actual user accessing the database for access control purposes.
- Diminished user accountability

  Accountability through auditing is a basic principle in database security. Not knowing the user's identity makes it difficult to distinguish the transactions performed by the middle tier for its own purpose from those performed by the middle tier on behalf of a user.
- Over granting of privileges to the middle tier's authorization ID

  The middle tier's authorization ID must have all the privileges necessary to execute all the requests from all the users. This has the security issue of enabling users who do not need access to certain information to obtain access anyway.
- Weakened security

  In addition to the privilege issue raised in the previous point, the current approach requires that the authorization ID used by the middle tier to connect

must be granted privileges on all resources that might be accessed by user requests. If that middle-tier authorization ID is ever compromised, then all those resources will be exposed.

- ″Spill over″ between users of the same connection

  Changes by a previous user can affect the current user.

Clearly, there is a need for a mechanism whereby the actual user's identity and database privileges are used for database requests performed by the middle tier on behalf of that user. The most straightforward approach of achieving this goal would be for the middle-tier to establish a new connection using the user's ID and password, and then direct the user's requests through that connection. Although simple, this approach suffers from several drawbacks which include the following:

- Inapplicability for certain middle tiers. Many middle-tier servers do not have the user authentication credentials needed to establish a connection.
- Performance overhead. There is an obvious performance overhead associated with creating a new physical connection and re-authenticating the user at the database server.
- Maintenance overhead. In situations where you are not using a centralized security set up or are not using single sign-on, there is maintenance overhead in having two user definitions (one on the middle tier and one at the server). This requires changing passwords at different places.

The trusted contexts capability addresses this problem. The security administrator can create a trusted context object in the database that defines a trust relationship between the database and the middle-tier. The middle-tier can then establish an explicit trusted connection to the database, which gives the middle tier the ability to switch the current user ID on the connection to a different user ID, with or without authentication. In addition to solving the end-user identity assertion problem, trusted contexts offer another advantage. This is the ability to control when a privilege is made available to a database user. The lack of control on when privileges are available to a user can weaken overall security. For example, privileges may be used for purposes other than they were originally intended. The security administrator can assign one or more privileges to a role and assign that role to a trusted context object. Only trusted database connections (explicit or implicit) that match the definition of that trusted context can take advantage of the privileges associated with that role.

Enhancing performance

When you use trusted connections, you can maximize performance because of the following advantages:

- No new connection is established when the current user ID of the connection is switched.
- If the trusted context definition does not require authentication of the user ID to switch to, then the overhead associated with authenticating a new user at the database server is not incurred.

Example of creating a trusted context

Suppose that the security administrator creates the following trusted context object:

```
CREATE TRUSTED CONTEXT CTX1
  BASED UPON CONNECTION USING SYSTEM AUTHID USER2
  ATTRIBUTES (ADDRESS   '192.0.2.1')
  DEFAULT ROLE managerRole
   ENABLE
```

If user *user1* requests a trusted connection from IP address 192.0.2.1, the DB2 database system returns a warning (SQLSTATE 01679, SQLCODE +20360) to indicate that a trusted connection could not be established, and that user *user1* simply got a non-trusted connection. However, if user *user2* requests a trusted connection from IP address 192.0.2.1, the request is honored because the connection attributes are satisfied by the trusted context CTX1. Now that use *user2* has established a trusted connection, he or she can now acquire all the privileges and authorities associated with the trusted context role managerRole. These privileges and authorities may not be available to user *user2* outside the scope of this trusted connection

Role membership inheritance through a trusted context

The current user of a trusted connection can acquire additional privileges through the automatic inheritance of a role through the trusted context, if this was specified by the security administrator as part of the relevant trusted context definition.

A role can be inherited by all users of the trusted connection by default. The security administrator can also use the trusted context definition to specify a role for specific users to inherit.

The active roles that a session authorization ID can hold while on a trusted connection are:

- The roles of which the session authorization ID is normally considered a member, plus
- Either the trusted context default role or the trusted context user-specific role, if they are defined

Note:

- If you configure user authentication using a custom security plugin that is built such that the system authorization ID and the session authorization ID produced by this security plugin upon a successful connection are different from each other, then a trusted contexts role cannot be inherited through that connection, even if it is a trusted connection.
- Trusted context privileges acquired through a role are effective only for dynamic DML operations. They are not effective for:
  - DDL operations
  - Non-dynamic SQL (operations involving static SQL statements such as BIND, REBIND, implicit rebind, incremental bind, and so on)

Acquiring trusted context user-specific privileges

The security administrator can use the trusted context definition to associate roles with a trusted context so that:

- All users of the trusted connection can inherit a specified role by default
- Specific users of the trusted connection can inherit a specified role

When the user on a trusted connection is switched to a new authorization ID and a trusted context user-specific role exists for this new authorization ID, the user-specific role overrides the trusted context default role, if one exists, as demonstrated in the example.

Example of creating a trusted context that assigns a default role and a user-specific role

Suppose that the security administrator creates the following trusted context object:

```
CREATE TRUSTED CONTEXT CTX1
  BASED UPON CONNECTION USING SYSTEM AUTHID USER1
 ATTRIBUTES (ADDRESS   '192.0.2.1')
  WITH USE FOR USER2 WITH AUTHENTICATION,
               USER3 WITHOUT AUTHENTICATION
  DEFAULT ROLE AUDITOR
  ENABLE
```

When USER1 establishes a trusted connection, the privileges granted to the role AUDITOR are inherited by this authorization ID. Similarly, these same privileges are also inherited by USER3 when the current authorization ID on the trusted connection is switched to his or her user ID. (If the user ID of the connection is switched to USER2 at some point, then USER2 would also inherit the trusted context default role, AUDITOR.) The security administrator may choose to have USER3 inherit a different role than the trusted context default role. They can do so by assigning a specific role to this user as follows:

```
CREATE TRUSTED CONTEXT CTX1
 BASED UPON CONNECTION USING SYSTEM AUTHID USER1
 ATTRIBUTES (ADDRESS   '192.0.2.1')
  WITH USE FOR USER2 WITH AUTHENTICATION,
               USER3 WITHOUT AUTHENTICATION ROLE OTHER_ROLE
  DEFAULT ROLE AUDITOR
  ENABLE
```

When the current user ID on the trusted connection is switched to USER3, this user no longer inherits the trusted context default role. Rather, they inherit the specific role, OTHER_ROLE, assigned to him or her by the security administrator.

Rules for switching the user ID on an explicit trusted connection

On an explicit trusted connection, you can switch the user ID of the connection to a different user ID. Certain rules apply.

1. If the switch request is not made from an explicit trusted connection, and the switch request is sent to the server for processing, the connection is shut down and an error message is returned (SQLSTATE 08001, SQLCODE -30082 with reason code 41).
2. If the switch request is not made on a transaction boundary, the transaction is rolled back, and the switch request is sent to the server for processing, the connection is put into an unconnected state and an error message is returned (SQLSTATE 58009, SQLCODE -30020).
3. If the switch request is made from within a stored procedure, an error message is returned (SQLCODE -30090, reason code 29), indicating this is an illegal operation in this environment. The connection state is maintained and the connection is not placed into an unconnected state. Subsequent requests may be processed.
4. If the switch request is delivered to the server on an instance attach (rather than a database connection), the attachment is shut down and an error message is returned (SQLCODE -30005).
5. If the switch request is made with an authorization ID that is not allowed on the trusted connection, error (SQLSTATE 42517, SQLCODE -20361) is returned, and the connection is put in an unconnected state.

6. If the switch request is made with an authorization ID that is allowed on the trusted connection WITH AUTHENTICATION, but the appropriate authentication token is not provided, error (SQLSTATE 42517, SQLCODE -20361) is returned, and the connection is put in an unconnected state.
7. If the trusted context object associated with the trusted connection is disabled, and a switch request for that trusted connection is made, error (SQLSTATE 42517, SQLCODE -20361) is returned, and the connection is put in an unconnected state.

   In this case, the only switch user request that is accepted is one that specifies the user ID that established the trusted connection or the NULL user ID. If a switch to the user ID that established the trusted connection is made, this user ID does not inherit any trusted context role (neither the trusted context default role nor the trusted context user-specific role).
8. If the system authorization ID attribute of the trusted context object associated with the trusted connection is changed, and a switch request for that trusted connection is made, error (SQLSTATE 42517, SQLCODE -20361) is returned, and the connection is put in an unconnected state.

   In this case, the only switch user request that is accepted is one that specifies the user ID that established the trusted connection or the NULL user ID. If a switch to the user ID that established the trusted connection is made, this user ID does not inherit any trusted context role (neither the trusted context default role nor the trusted context user-specific role).
9. If the trusted context object associated with the trusted connection is dropped, and a switch request for that trusted connection is made, error (SQLSTATE 42517, SQLCODE -20361) is returned, and the connection is put in an unconnected state.

   In this case, the only switch user request that is accepted is one that specifies the user ID that established the trusted connection or the NULL user ID. If a switch to the user ID that established the trusted connection is made, this user ID does not inherit any trusted context role (neither the trusted context default role nor the trusted context user-specific role).
10. If the switch request is made with a user ID allowed on the trusted connection, but that user ID does not hold CONNECT privilege on the database, the connection is put in an unconnected state and an error message is returned (SQLSTATE 08004, SQLCODE -1060).
11. If the trusted context system authorization ID appears in the WITH USE FOR clause, the DB2 database system honors the authentication setting for the system authorization ID on switch user request to switch back to the system authorization ID. If the trusted context system authorization ID does not appear in the WITH USE FOR clause, then a switch user request to switch back to the system authorization ID is always allowed even without authentication.

Note: When the connection is put in the unconnected state, the only requests that are accepted and do not result in returning the error ″The application state is in error. A database connection does not exist.″ (SQLCODE -900) are:

- A switch user request
- A COMMIT or ROLLBACK statement
- A DISCONNECT, CONNECT RESET or CONNECT request

Note: When the user ID on the trusted connection is switched to a new user ID, all traces of the connection environment under the old user are gone. In other words, the switching of user IDs results in an environment that is identical to a

new connection environment. For example, if the old user ID on the connection had any temporary tables or WITH HOLD cursors open, these objects are completely lost when the user ID on that connection is switched to a new user ID.

Trusted context problem determination

An explicit trusted connection is a connection that is successfully established by a specific, explicit request for a trusted connection. When you request an explicit trusted connection and you do not qualify for one, you get a regular connection and a warning (+20360).

To determine why a user could not establish a trusted connection, the security administrator needs to look at the trusted context definition in the system catalogs and at the connection attributes. In particular, the IP address from which the connection is established, the encryption level of the data stream or network, and the system authorization ID making the connection. The -application option of the db2pd utility returns this information, as well as the following additional information:

- Connection Trust Type: Indicates whether the connection is trusted or not. When the connection is trusted, this also indicates whether this is an explicit trusted connection or an implicit trusted connection.
- Trusted Context name: The name of the trusted context associated with the trusted connection.
- Role Inherited: The role inherited through the trusted connection.

The following are the most common causes of failing to obtain an explicit trusted connection:

- The client application is not using TCP/IP to communicate with the DB2 server. TCP/IP is the only supported protocol for a client application to communicate with the DB2 server that can be used to establish a trusted connection (explicit or implicit).
- The database server authentication type is set to CLIENT.
- The database server does not have an enabled trusted context object. The definition of the trusted context object must explicitly state ENABLE in order for that trusted context to be considered for matching the attributes of an incoming connection.
- The trusted context objects on the database server do not match the trust attributes that are presented. For example, one of the following situations may apply:
  - The system authorization ID of the connection does not match any trusted context object system authorization ID.
  - The IP address from which the connection originated does not match any IP address in the trusted context object considered for the connection.
  - The data stream encryption used by the connection does not match the value of the ENCRYPTION attribute in the trusted context object considered for the connection.

  You can use the db2pd tool to find out the IP address from which the connection is established, the encryption level of the data stream or network used by the connection, and the system authorization ID making the connection. You can consult the SYSCAT.CONTEXTS and SYSCAT.CONTEXTATTRIBUTES catalog views to find out the definition of a particular trusted context object, such as its system authorization ID, its set of allowed IP addresses and the value of its ENCRYPTION attribute.

The following are the most common causes of a switch user failure:

- The user ID to switch to does not have CONNECT privileges on the database. In this case, SQL1060N is returned.
- The user ID to switch to, or PUBLIC, is not defined in the WITH USE FOR clause of the trusted context object associated with the explicit trusted connection.
- Switching the user is allowed with authentication, but the user presents no credentials or the wrong credentials.
- A switch-user request is not made on a transaction boundary.
- The trusted context that is associated with a trusted connection has been disabled, dropped, or altered. In this case, only switching to the user ID that established the trusted connection is allowed.

Chapter 24. Firewall considerations

Firewall support

A *firewall* is a set of related programs, located at a network gateway server, that are used to prevent unauthorized access to a system or network.

There are four types of firewalls:

1. Network level, packet-filter, or screening router firewalls
2. Classical application level proxy firewalls
3. Circuit level or transparent proxy firewalls
4. Stateful multi-layer inspection (SMLI) firewalls

There are existing firewall products that incorporate one of the firewall types listed above. There are many other firewall products that incorporate some combination of the above types.

Screening router firewalls

The screening router firewall is also known as a network level or packet-filter firewall. Such a firewall works by screening incoming packets by protocol attributes. The protocol attributes screened may include source or destination address, type of protocol, source or destination port, or some other protocol-specific attributes.

For all firewall solutions (except SOCKS), you need to ensure that all the ports used by DB2 database are open for incoming and outgoing packets. DB2 database uses port 523 for the DB2 Administration Server (DAS), which is used by the DB2 database tools. Determine the ports used by all your server instances by using the services file to map the service name in the server database manager configuration file to its port number.

Application proxy firewalls

A proxy or proxy server is a technique that acts as an intermediary between a Web client and a Web server. A proxy firewall acts as a gateway for requests arriving from clients.

When client requests are received at the firewall, the final server destination address is determined by the proxy software. The application proxy translates the address, performs additional access control checking and logging as necessary, and connects to the server on behalf of the client.

The DB2 Connect product on a firewall machine can act as a proxy to the destination server. Also, a DB2 database server on the firewall, acting as a hop server to the final destination server, acts like an application proxy.

Circuit level firewalls

The circuit level firewall is also known as a transparent proxy firewall.

A transparent proxy firewall does not modify the request or response beyond what is required for proxy authentication and identification. An example of a transparent proxy firewall is SOCKS.

The DB2 database system supports SOCKS Version 4.

Stateful multi-layer inspection (SMLI) firewalls

The stateful multi-layer inspection (SMLI) firewall uses a sophisticated form of packet-filtering that examines all seven layers of the Open System Interconnection (OSI) model.

Each packet is examined and compared against known states of friendly packets. While screening router firewalls only examine the packet header, SMLI firewalls examine the entire packet including the data.

Chapter 25. System catalogs and security maintenance

System catalogs and security maintenance

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.

Using the system catalog for security information

Information about each database is automatically maintained in a set of views called the system catalog, which is created when the database is created. This system catalog describes tables, columns, indexes, programs, privileges, and other objects.

The following views and table functions list information about privileges held by users, identities of users granting privileges, and object ownership:

SYSCAT.COLAUTH
Lists the column privileges

SYSCAT.DBAUTH
Lists the database privileges

SYSCAT.INDEXAUTH
Lists the index privileges

SYSCAT.MODULEAUTH
Lists the module privileges

SYSCAT.PACKAGEAUTH
Lists the package privileges

SYSCAT.PASSTHRUAUTH
Lists the server privilege

SYSCAT.ROLEAUTH
Lists the role privileges

SYSCAT.ROUTINEAUTH
Lists the routine (functions, methods, and stored procedures) privileges

SYSCAT.SCHEMAAUTH
Lists the schema privileges

SYSCAT.SEQUENCEAUTH
Lists the sequence privileges

SYSCAT.SURROGATEAUTHIDS
Lists the authorization IDs for which another authorization ID can act as a surrogate.

SYSCAT.TABAUTH
Lists the table and view privileges

SYSCAT.TBSPACEAUTH
Lists the table space privileges

SYSCAT.VARIABLEAUTH
Lists the variable privileges

SYSCAT.WORKLOADAUTH
Lists the workload privileges

SYSCAT.XSROBJECTAUTH
Lists the XSR object privileges

Privileges granted to users by the system will have SYSIBM as the grantor. SYSADM, SYSMAINT SYSCTRL, and SYSMON are not listed in the system catalog.

The CREATE and GRANT statements place privileges in the system catalog. Users with ACCESSCTRL and SECADM authority can grant and revoke SELECT privilege on the system catalog views.

Details on using the system catalog for security issues

Retrieving authorization names with granted privileges

You can use the PRIVILEGES and other administrative views to retrieve information about the authorization names that have been granted privileges in a database.

About this task

For example, the following query retrieves all explicit privileges and the authorization IDs to which they were granted, plus other information, from the PRIVILEGES administrative view:

```
SELECT AUTHID, PRIVILEGE, OBJECTNAME, OBJECTSCHEMA, OBJECTTYPE FROM SYSIBMADM.PRIVILEGES
```

The following query uses the AUTHORIZATIONIDS administrative view to find all the authorization IDs that have been granted privileges or authorities, and to show their types:

```
SELECT AUTHID, AUTHIDTYPE FROM SYSIBMADM.AUTHORIZATIONIDS
```

You can also use the SYSIBMADM.OBJECTOWNERS administrative view and the SYSPROC.AUTH_LIST_GROUPS_FOR_AUTHID table function to find security-related information.

Prior to Version 9.1, no single system catalog view contained information about all privileges. For releases earlier than version 9.1, the following statement retrieves all authorization names with privileges:

```
 SELECT DISTINCT GRANTEE, GRANTEETYPE, 'DATABASE' FROM SYSCAT.DBAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'TABLE   ' FROM SYSCAT.TABAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'PACKAGE ' FROM SYSCAT.PACKAGEAUTH
UNION
```

```
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'INDEX   ' FROM SYSCAT.INDEXAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'COLUMN  ' FROM SYSCAT.COLAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'SCHEMA  ' FROM SYSCAT.SCHEMAAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'SERVER  ' FROM SYSCAT.PASSTHRUAUTH
ORDER BY GRANTEE, GRANTEETYPE, 3
```

Periodically, the list retrieved by this statement should be compared with lists of user and group names defined in the system security facility. You can then identify those authorization names that are no longer valid.

Note: If you are supporting remote database clients, it is possible that the authorization name is defined at the remote client only and not on your database server machine.

Retrieving all names with DBADM authority

The following statement retrieves all authorization names that have been directly granted DBADM authority:

About this task

```
SELECT DISTINCT GRANTEE, GRANTEETYPE FROM SYSCAT.DBAUTH
   WHERE DBADMAUTH = 'Y'
```

Retrieving names authorized to access a table

You can use the PRIVILEGES and other administrative views to retrieve information about the authorization names that have been granted privileges in a database.

About this task

The following statement retrieves all authorization names (and their types) that are directly authorized to access the table EMPLOYEE with the qualifier JAMES:

```
SELECT DISTINCT AUTHID, AUTHIDTYPE FROM SYSIBMADM.PRIVILEGES
    WHERE OBJECTNAME = 'EMPLOYEE' AND OBJECTSCHEMA = 'JAMES'
```

For releases earlier than Version 9.1, the following query retrieves the same information:

```
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.TABAUTH
   WHERE TABNAME = 'EMPLOYEE'
     AND TABSCHEMA = 'JAMES'
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.COLAUTH
   WHERE TABNAME = 'EMPLOYEE'
     AND TABSCHEMA = 'JAMES'
```

To find out who can update the table EMPLOYEE with the qualifier JAMES, issue the following statement:

```
 SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.TABAUTH
   WHERE TABNAME = 'EMPLOYEE' AND TABSCHEMA = 'JAMES' AND
      (CONTROLAUTH  = 'Y' OR
       UPDATEAUTH IN ('G','Y'))
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.DBAUTH
   WHERE DBADMAUTH = 'Y'
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.COLAUTH
   WHERE TABNAME = 'EMPLOYEE' AND TABSCHEMA = 'JAMES' AND
   PRIVTYPE = 'U'
```

This retrieves any authorization names with DBADM authority, as well as those names to which CONTROL or UPDATE privileges have been directly granted.

Remember that some of the authorization names may be groups, not just individual users.

Retrieving all privileges granted to users

By making queries on the system catalog views, users can retrieve a list of the privileges they hold and a list of the privileges they have granted to other users.

About this task

You can use the PRIVILEGES and other administrative views to retrieve information about the authorization names that have been granted privileges in a database. For example, the following query retrieves all the privileges granted to the current session authorization ID:

```
SELECT * FROM SYSIBMADM.PRIVILEGES
 WHERE AUTHID = SESSION_USER AND AUTHIDTYPE = 'U'
```

The keyword SESSION_USER in this statement is a special register that is equal to the value of the current user's authorization name.

For releases earlier than Version 9.1, the following examples provide similar information. For example, the following statement retrieves a list of the database privileges that have been directly granted to the individual authorization name JAMES:

```
SELECT * FROM SYSCAT.DBAUTH
 WHERE GRANTEE = 'JAMES' AND GRANTEETYPE = 'U'
```

The following statement retrieves a list of the table privileges that were directly granted by the user JAMES:

```
SELECT * FROM SYSCAT.TABAUTH
 WHERE GRANTOR  = 'JAMES'
```

The following statement retrieves a list of the individual column privileges that were directly granted by the user JAMES:

```
SELECT * FROM SYSCAT.COLAUTH
   WHERE GRANTOR  = 'JAMES'
```

Securing the system catalog view

Because the system catalog views describe every object in the database, if you have sensitive data, you might want to restrict their access.

About this task

The following authorities have SELECT privilege on all catalog tables:

- ACCESSCTRL
- DATAACCESS
- DBADM
- SECADM
- SQLADM

In addition, the following instance level authorities have the ability to select from SYSCAT.BUFFERPOOLS, SYSCAT.DBPARTITIONGROUPS, SYSCAT.DBPARTITIONGROUPDEF, SYSCAT.PACKAGES, and SYSCAT.TABLES:

- SYSADM
- SYSCTRL
- SYSMAINT
- SYSMON

You can use the CREATE DATABASE ... RESTRICTIVE command to create a database in which no privileges are automatically granted to PUBLIC. In this case, none of the following normal default grant actions occur:

- CREATETAB
- BINDADD
- CONNECT
- IMPLICIT_SCHEMA
- EXECUTE with GRANT on all procedures in schema SQLJ
- EXECUTE with GRANT on all functions and procedures in schema SYSPROC
- BIND on all packages created in the NULLID schema
- EXECUTE on all packages created in the NULLID schema
- CREATEIN on schema SQLJ
- CREATEIN on schema NULLID
- USE on table space USERSPACE1
- SELECT access to the SYSIBM catalog tables
- SELECT access to the SYSCAT catalog views
- SELECT access to the SYSIBMADM administrative views
- SELECT access to the SYSSTAT catalog views
- UPDATE access to the SYSSTAT catalog views

If you have created a database using the RESTRICTIVE option, and you want to check that the permissions granted to PUBLIC are limited, you can issue the following query to verify which schemas PUBLIC can access:

```
SELECT DISTINCT OBJECTSCHEMA FROM SYSIBMADM.PRIVILEGES WHERE AUTHID='PUBLIC'

OBJECTSCHEMA
------------
SYSFUN
SYSIBM
SYSPROC
```

To see what access PUBLIC still has to SYSIBM, you can issue the following query to check what privileges are granted on SYSIBM. The results show that only EXECUTE on certain procedures and functions is granted.

```
SELECT * FROM SYSIBMADM.PRIVILEGES WHERE OBJECTSCHEMA = 'SYSIBM'

AUTHID       AUTHIDTYPE  PRIVILEGE   GRANTABLE  OBJECTNAME         OBJECTSCHEMA    OBJECTTYPE
---------... ----------  ----------  ---------  ---------------... ------------... ----------
PUBLIC       G           EXECUTE     N          SQL060207192129400 SYSPROC         FUNCTION
PUBLIC       G           EXECUTE     N          SQL060207192129700 SYSPROC         FUNCTION
PUBLIC       G           EXECUTE     N          SQL060207192129701 SYSPROC
...
PUBLIC       G           EXECUTE     Y          TABLES             SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          TABLEPRIVILEGES    SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          STATISTICS         SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          SPECIALCOLUMNS     SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          PROCEDURES         SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          PROCEDURECOLS      SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          PRIMARYKEYS        SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          FOREIGNKEYS        SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          COLUMNS            SYSIBM          PROCEDURE
```

```
PUBLIC      G           EXECUTE     Y          COLPRIVILEGES      SYSIBM          PROCEDURE
PUBLIC      G           EXECUTE     Y          UDTS               SYSIBM          PROCEDURE
PUBLIC      G           EXECUTE     Y          GETTYPEINFO        SYSIBM          PROCEDURE
PUBLIC      G           EXECUTE     Y          SQLCAMESSAGE       SYSIBM          PROCEDURE
PUBLIC      G           EXECUTE     Y          SQLCAMESSAGECCSID  SYSIBM          PROCEDURE
```

Note: The SYSIBMADM.PRIVILEGES administrative view is available starting with Version 9.1 of the DB2 database manager.

For releases earlier than Version 9.1 of the DB2 database manager, during database creation, SELECT privilege on the system catalog views is granted to PUBLIC. In most cases, this does not present any security problems. For very sensitive data, however, it may be inappropriate, as these tables describe every object in the database. If this is the case, consider revoking the SELECT privilege from PUBLIC; then grant the SELECT privilege as required to specific users. Granting and revoking SELECT on the system catalog views is done in the same way as for any view, but you must have either ACCESSCTRL or SECADM authority to do this.

At a minimum, if you don't want any user to be able to know what objects other users have access to, you should consider restricting access to the following catalog and administrative views:

- SYSCAT.COLAUTH
- SYSCAT.DBAUTH
- SYSCAT.INDEXAUTH
- SYSCAT.PACKAGEAUTH
- SYSCAT.PASSTHRUAUTH
- SYSCAT.ROUTINEAUTH
- SYSCAT.SCHEMAAUTH
- SYSCAT.SECURITYLABELACCESS
- SYSCAT.SECURITYPOLICYEXEMPTIONS
- SYSCAT.SEQUENCEAUTH
- SYSCAT.SURROGATEAUTHIDS
- SYSCAT.TABAUTH
- SYSCAT.TBSPACEAUTH
- SYSCAT.XSROBJECTAUTH
- SYSIBMADM.AUTHORIZATIONIDS
- SYSIBMADM.OBJECTOWNERS
- SYSIBMADM.PRIVILEGES

This would prevent information on user privileges from becoming available to everyone with access to the database.

You should also examine the columns for which statistics are gathered. Some of the statistics recorded in the system catalog contain data values which could be sensitive information in your environment. If these statistics contain sensitive data, you may wish to revoke SELECT privilege from PUBLIC for the SYSCAT.COLUMNS and SYSCAT.COLDIST catalog views.

If you wish to limit access to the system catalog views, you could define views to let each authorization name retrieve information about its own privileges.

For example, the following view MYSELECTS includes the owner and name of every table on which a user's authorization name has been directly granted SELECT privilege:

```
CREATE VIEW MYSELECTS AS
   SELECT TABSCHEMA, TABNAME FROM SYSCAT.TABAUTH
   WHERE GRANTEETYPE = 'U'
     AND GRANTEE = USER
     AND SELECTAUTH = 'Y'
```

The keyword USER in this statement is equal to the value of the current session authorization name.

The following statement makes the view available to every authorization name:

```
GRANT SELECT ON TABLE MYSELECTS TO PUBLIC
```

And finally, remember to revoke SELECT privilege on the view and base table by issuing the following two statements:

```
REVOKE SELECT ON TABLE SYSCAT.TABAUTH FROM PUBLIC
REVOKE SELECT ON TABLE SYSIBM.SYSTABAUTH FROM PUBLIC
```

System catalog views

System catalog views

The database manager creates and maintains two sets of system catalog views that are defined on top of the base system catalog tables.

- SYSCAT views are read-only catalog views that are found in the SYSCAT schema. The RESTRICT option on CREATE DATABASE statement determines how SELECT privilege is granted. When the RESTRICT option is not specified, SELECT privilege is granted to PUBLIC.
- SYSSTAT views are updatable catalog views that are found in the SYSSTAT schema. The updatable views contain statistical information that is used by the optimizer. The values in some columns in these views can be changed to test performance. (Before changing any statistics, it is recommended that the RUNSTATS command be invoked so that all the statistics reflect the current state.)

Applications should be written to the SYSCAT and SYSSTAT views rather than the base catalog tables.

All the system catalog views are created at database creation time. The catalog views cannot be explicitly created or dropped. In a Unicode database, the catalog views are created with IDENTITY collation. In non-Unicode databases, the catalog views are created with the database collation. The views are updated during normal operation in response to SQL data definition statements, environment routines, and certain utilities. Data in the system catalog views is available through normal SQL query facilities. The system catalog views (with the exception of some updatable catalog views) cannot be modified using normal SQL data manipulation statements.

A object table, column, or index object will appear in a user's updatable SYSSTAT catalog view only if that user holds CONTROL privilege on the object, or holds explicit DATAACCESS authority. A routine object will appear in a user's updatable SYSSTAT.ROUTINES catalog view if that user owns the routine or holds SQLADM authority.

The order of columns in the views may change from release to release. To prevent this from affecting programming logic, specify the columns in a select list explicitly, and avoid using `SELECT *`. Columns have consistent names based on the types of objects that they describe.

Table 35. Samples of consistent column names for objects they descibe

Described Object	**Column Names**
Table	TABSCHEMA, TABNAME
Index	INDSCHEMA, INDNAME
Index extension	IESCHEMA, IENAME
View	VIEWSCHEMA, VIEWNAME
Constraint	CONSTSCHEMA, CONSTNAME
Trigger	TRIGSCHEMA, TRIGNAME
Package	PKGSCHEMA, PKGNAME
Type	TYPESCHEMA, TYPENAME, TYPEID
Function	ROUTINESCHEMA, ROUTINEMODULENAME, ROUTINENAME, ROUTINEID
Method	ROUTINESCHEMA, ROUTINEMODULENAME, ROUTINENAME, ROUTINEID
Procedure	ROUTINESCHEMA, ROUTINEMODULENAME, ROUTINENAME, ROUTINEID
Column	COLNAME
Schema	SCHEMANAME
Table Space	TBSPACE
Database partition group	DBPGNAME
Audit policy	AUDITPOLICYNAME, AUDITPOLICYID
Buffer pool	BPNAME
Event Monitor	EVMONNAME
Condition	CONDSCHEMA, CONDMODULENAME, CONDNAME, CONDMODULEID
Data source	SERVERNAME, SERVERTYPE, SERVERVERSION
Global variable	VARSCHEMA, VARMODULENAME, VARNAME, VARMODULEID
Histogram template	TEMPLATENAME, TEMPLATEID
Module	MODULESCHEMA, MODULENAME, MODULEID
Role	ROLENAME, ROLEID
Security label	SECLABELNAME, SECLABELID
Security policy	SECPOLICYNAME, SECPOLICYID
Sequence	SEQSCHEMA, SEQNAME
Threshold	THRESHOLDNAME, THRESHOLDID
Trusted context	CONTEXTNAME, CONTEXTID
Work action	ACTIONNAME, ACTIONID
Work action set	ACTIONSETNAME, ACTIONSETID
Work class	WORKCLASSNAME, WORKCLASSID

Table 35. Samples of consistent column names for objects they descibe (continued)

Described Object	Column Names
Work class set	WORKCLASSSETNAME, WORKCLASSSETID
Workload	WORKLOADID, WORKLOADNAME
Wrapper	WRAPNAME
Alteration Timestamp	ALTER_TIME
Creation Timestamp	CREATE_TIME

Road map to the catalog views

Table 36. Road map to the read-only catalog views

Description	Catalog View
attributes of structured data types	"SYSCAT.ATTRIBUTES" on page 947
audit policies	"SYSCAT.AUDITPOLICIES" on page 221
	"SYSCAT.AUDITUSE" on page 222
authorities on database	"SYSCAT.DBAUTH" on page 224
buffer pool configuration on database partition group	"SYSCAT.BUFFERPOOLS" on page 951
buffer pool size on database partition	"SYSCAT.BUFFERPOOLDBPARTITIONS" on page 950
cast functions	"SYSCAT.CASTFUNCTIONS" on page 951
check constraints	"SYSCAT.CHECKS" on page 952
column privileges	"SYSCAT.COLAUTH" on page 223
columns	"SYSCAT.COLUMNS" on page 957
columns referenced by check constraints	"SYSCAT.COLCHECKS" on page 953
columns used in dimensions	"SYSCAT.COLUSE" on page 962
columns used in keys	"SYSCAT.KEYCOLUSE" on page 994
conditions	"SYSCAT.CONDITIONS" on page 963
constraint dependencies	"SYSCAT.CONSTDEP" on page 963
database partition group database partitions	"SYSCAT.DBPARTITIONGROUPDEF" on page 971
database partition group definitions	"SYSCAT.DBPARTITIONGROUPS" on page 971
data partitions	"SYSCAT.DATAPARTITIONEXPRESSION" on page 964
	"SYSCAT.DATAPARTITIONS" on page 965
data type dependencies	"SYSCAT.DATATYPEDEP" on page 967
data types	"SYSCAT.DATATYPES" on page 968
detailed column group statistics	"SYSCAT.COLGROUPCOLS" on page 955
	"SYSCAT.COLGROUPDIST" on page 955
	"SYSCAT.COLGROUPDISTCOUNTS" on page 955
	"SYSCAT.COLGROUPS" on page 956
detailed column options	"SYSCAT.COLOPTIONS" on page 957
detailed column statistics	"SYSCAT.COLDIST" on page 954
distribution maps	"SYSCAT.PARTITIONMAPS" on page 1004
event monitor definitions	"SYSCAT.EVENTMONITORS" on page 972

Table 36. Road map to the read-only catalog views (continued)

Description	Catalog View
events currently monitored	"SYSCAT.EVENTS" on page 973
	"SYSCAT.EVENTTABLES" on page 974
fields of row data types	"SYSCAT.ROWFIELDS" on page 1009
function dependencies[1]	"SYSCAT.ROUTINEDEP" on page 1007
function mapping	"SYSCAT.FUNCMAPPINGS" on page 976
function mapping options	"SYSCAT.FUNCMAPOPTIONS" on page 976
function parameter mapping options	"SYSCAT.FUNCMAPPARMOPTIONS" on page 976
function parameters[1]	"SYSCAT.ROUTINEPARMS" on page 1010
functions[1]	"SYSCAT.ROUTINES" on page 1012
global variables	"SYSCAT.VARIABLEAUTH" on page 246
	"SYSCAT.VARIABLEDEP" on page 1034
	"SYSCAT.VARIABLES" on page 1035
hierarchies (types, tables, views)	"SYSCAT.HIERARCHIES" on page 977
	"SYSCAT.FULLHIERARCHIES" on page 975
identity columns	"SYSCAT.COLIDENTATTRIBUTES" on page 956
index columns	"SYSCAT.INDEXCOLUSE" on page 979
index data partitions	"SYSCAT.INDEXPARTITIONS" on page 990
index dependencies	"SYSCAT.INDEXDEP" on page 980
index exploitation	"SYSCAT.INDEXEXPLOITRULES" on page 986
index extension dependencies	"SYSCAT.INDEXEXTENSIONDEP" on page 987
index extension parameters	"SYSCAT.INDEXEXTENSIONPARMS" on page 988
index extension search methods	"SYSCAT.INDEXEXTENSIONMETHODS" on page 988
index extensions	"SYSCAT.INDEXEXTENSIONS" on page 989
index options	"SYSCAT.INDEXOPTIONS" on page 990
index privileges	"SYSCAT.INDEXAUTH" on page 225
indexes	"SYSCAT.INDEXES" on page 981
invalid objects	"SYSCAT.INVALIDOBJECTS" on page 993
method dependencies[1]	"SYSCAT.ROUTINEDEP" on page 1007
method parameters[1]	"SYSCAT.ROUTINES" on page 1012
methods[1]	"SYSCAT.ROUTINES" on page 1012
module objects	"SYSCAT.MODULEOBJECTS" on page 995
module privileges	"SYSCAT.MODULEAUTH" on page 994
modules	"SYSCAT.MODULES" on page 996
nicknames	"SYSCAT.NICKNAMES" on page 997
object mapping	"SYSCAT.NAMEMAPPINGS" on page 996
package dependencies	"SYSCAT.PACKAGEDEP" on page 226
package privileges	"SYSCAT.PACKAGEAUTH" on page 226
packages	"SYSCAT.PACKAGES" on page 1000
partitioned tables	"SYSCAT.TABDETACHEDDEP" on page 1025
pass-through privileges	"SYSCAT.PASSTHRUAUTH" on page 228

Table 36. Road map to the read-only catalog views (continued)

Description	Catalog View
predicate specifications	"SYSCAT.PREDICATESPECS" on page 1005
procedure options	"SYSCAT.ROUTINEOPTIONS" on page 1008
procedure parameter options	"SYSCAT.ROUTINEPARMOPTIONS" on page 1010
procedure parameters[1]	"SYSCAT.ROUTINEPARMS" on page 1010
procedures[1]	"SYSCAT.ROUTINES" on page 1012
protected tables	"SYSCAT.SECURITYLABELACCESS" on page 232
	"SYSCAT.SECURITYLABELCOMPONENTELEMENTS" on page 232
	"SYSCAT.SECURITYLABELCOMPONENTS" on page 233
	"SYSCAT.SECURITYLABELS" on page 233
	"SYSCAT.SECURITYPOLICIES" on page 233
	"SYSCAT.SECURITYPOLICYCOMPONENTRULES" on page 234
	"SYSCAT.SECURITYPOLICYEXEMPTIONS" on page 235
	"SYSCAT.SURROGATEAUTHIDS" on page 235
provides DB2 for z/OS compatibility	"SYSIBM.SYSDUMMY1" on page 1050
referential constraints	"SYSCAT.REFERENCES" on page 1005
remote table options	"SYSCAT.TABOPTIONS" on page 1025
roles	"SYSCAT.ROLEAUTH" on page 228
	"SYSCAT.ROLES" on page 229
routine dependencies	"SYSCAT.ROUTINEDEP" on page 1007
routine parameters[1]	"SYSCAT.ROUTINEPARMS" on page 1010
routine privileges	"SYSCAT.ROUTINEAUTH" on page 230
routines[1]	"SYSCAT.ROUTINES" on page 1012
	"SYSCAT.ROUTINESFEDERATED" on page 1019
schema privileges	"SYSCAT.SCHEMAAUTH" on page 229
schemas	"SYSCAT.SCHEMATA" on page 231
sequence privileges	"SYSCAT.SEQUENCEAUTH" on page 231
sequences	"SYSCAT.SEQUENCES" on page 236
server options	"SYSCAT.SERVEROPTIONS" on page 1021
server-specific user options	"SYSCAT.USEROPTIONS" on page 246
statements in packages	"SYSCAT.STATEMENTS" on page 1023
stored procedures	"SYSCAT.ROUTINES" on page 1012
system servers	"SYSCAT.SERVERS" on page 1021
table constraints	"SYSCAT.TABCONST" on page 238
table dependencies	"SYSCAT.TABDEP" on page 1023
table privileges	"SYSCAT.TABAUTH" on page 247
table space use privileges	"SYSCAT.TBSPACEAUTH" on page 245
table spaces	"SYSCAT.TABLESPACES" on page 244
tables	"SYSCAT.TABLES" on page 239

Table 36. Road map to the read-only catalog views (continued)

Description	Catalog View
transforms	"SYSCAT.TRANSFORMS" on page 1027
trigger dependencies	"SYSCAT.TRIGDEP" on page 1028
triggers	"SYSCAT.TRIGGERS" on page 1029
trusted contexts	"SYSCAT.CONTEXTATTRIBUTES" on page 964
	"SYSCAT.CONTEXTS" on page 964
type mapping	"SYSCAT.TYPEMAPPINGS" on page 1031
user-defined functions	"SYSCAT.ROUTINES" on page 1012
view dependencies	"SYSCAT.TABDEP" on page 1023
views	"SYSCAT.TABLES" on page 239
	"SYSCAT.VIEWS" on page 1037
workload management	"SYSCAT.HISTOGRAMTEMPLATEBINS" on page 978
	"SYSCAT.HISTOGRAMTEMPLATES" on page 978
	"SYSCAT.HISTOGRAMTEMPLATEUSE" on page 978
	"SYSCAT.SERVICECLASSES" on page 1021
	"SYSCAT.THRESHOLDS" on page 1025
	"SYSCAT.WORKACTIONS" on page 1037
	"SYSCAT.WORKACTIONSETS" on page 1040
	"SYSCAT.WORKCLASSES" on page 1040
	"SYSCAT.WORKCLASSSETS" on page 1041
	"SYSCAT.WORKLOADAUTH" on page 1042
	"SYSCAT.WORKLOADCONNATTR" on page 1042
	"SYSCAT.WORKLOADS" on page 1043
wrapper options	"SYSCAT.WRAPOPTIONS" on page 1045
wrappers	"SYSCAT.WRAPPERS" on page 1045
XML strings	"SYSCAT.XMLSTRINGS" on page 1046
XML values index	"SYSCAT.INDEXXMLPATTERNS" on page 993
XSR objects	"SYSCAT.XDBMAPGRAPHS" on page 1045
	"SYSCAT.XDBMAPSHREDTREES" on page 1046
	"SYSCAT.XSROBJECTAUTH" on page 1046
	"SYSCAT.XSROBJECTCOMPONENTS" on page 1047
	"SYSCAT.XSROBJECTDEP" on page 1048
	"SYSCAT.XSROBJECTDETAILS" on page 1047
	"SYSCAT.XSROBJECTHIERARCHIES" on page 1049
	"SYSCAT.XSROBJECTS" on page 1049

[1] The following catalog views for functions, methods, and procedures defined in DB2 Version 7.1 and earlier are still available:

```
Functions:  SYSCAT.FUNCTIONS, SYSCAT.FUNCDEP, SYSCAT.FUNCPARMS
Methods:    SYSCAT.FUNCTIONS, SYSCAT.FUNCDEP, SYSCAT.FUNCPARMS
Procedures: SYSCAT.PROCEDURES, SYSCAT.PROCPARMS
```

However, these views have not been updated since DB2 Version 7.1. Use the SYSCAT.ROUTINES, SYSCAT.ROUTINEDEP, or SYSCAT.ROUTINEPARMS catalog view instead.

Table 37. Road map to the updatable catalog views

Description	Catalog View
columns	"SYSSTAT.COLUMNS" on page 1052
detailed column group statistics	"SYSSTAT.COLGROUPDIST" on page 1051
	"SYSSTAT.COLGROUPDISTCOUNTS" on page 1052
	"SYSSTAT.COLGROUPS" on page 1052
detailed column statistics	"SYSSTAT.COLDIST" on page 1050
indexes	"SYSSTAT.INDEXES" on page 1054
routines[1]	"SYSSTAT.ROUTINES" on page 1057
tables	"SYSSTAT.TABLES" on page 1058

[1] The SYSSTAT.FUNCTIONS catalog view still exists for updating the statistics for functions and methods. This view, however, does not reflect any changes since DB2 Version 7.1.

SYSCAT.AUDITPOLICIES

Each row represents an audit policy.

Table 38. SYSCAT.AUDITPOLICIES Catalog View

Column Name	Data Type	Nullable	Description
AUDITPOLICYNAME	VARCHAR (128)		Name of the audit policy.
AUDITPOLICYID	INTEGER		Identifier for the audit policy.
CREATE_TIME	TIMESTAMP		Time at which the audit policy was created.
ALTER_TIME	TIMESTAMP		Time at which the audit policy was last altered.
AUDITSTATUS	CHAR (1)		Status for the AUDIT category. • B = Both • F = Failure • N = None • S = Success
CONTEXTSTATUS	CHAR (1)		Status for the CONTEXT category. • B = Both • F = Failure • N = None • S = Success
VALIDATESTATUS	CHAR (1)		Status for the VALIDATE category. • B = Both • F = Failure • N = None • S = Success
CHECKINGSTATUS	CHAR (1)		Status for the CHECKING category. • B = Both • F = Failure • N = None • S = Success

Table 38. SYSCAT.AUDITPOLICIES Catalog View (continued)

Column Name	Data Type	Nullable	Description
SECMAINTSTATUS	CHAR (1)		Status for the SECMAINT category. • B = Both • F = Failure • N = None • S = Success
OBJMAINTSTATUS	CHAR (1)		Status for the OBJMAINT category. • B = Both • F = Failure • N = None • S = Success
SYSADMINSTATUS	CHAR (1)		Status for the SYSADMIN category. • B = Both • F = Failure • N = None • S = Success
EXECUTESTATUS	CHAR (1)		Status for the EXECUTE category. • B = Both • F = Failure • N = None • S = Success
EXECUTEWITHDATA	CHAR (1)		Host variables and parameter markers logged with EXECUTE category. • N = No • Y = Yes
ERRORTYPE	CHAR (1)		The audit error type. • A = Audit • N = Normal
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.AUDITUSE

Each row represents an audit policy that is associated with a non-database object, such as USER, GROUP, or authority (SYSADM, SYSCTRL, SYSMAINT).

Table 39. SYSCAT.AUDITUSE Catalog View

Column Name	Data Type	Nullable	Description
AUDITPOLICYNAME	VARCHAR (128)		Name of the audit policy.
AUDITPOLICYID	INTEGER		Identifier for the audit policy.

Table 39. SYSCAT.AUDITUSE Catalog View (continued)

Column Name	Data Type	Nullable	Description
OBJECTTYPE	CHAR(1)		The type of object with which this audit policy is associated. • S = MQT • T = Table • g = Authority • i = Authorization ID • x = Trusted context • Blank = Database
SUBOBJECTTYPE	CHAR(1)		If OBJECTTYPE is 'i', this is the type that the authorization ID represents. • G = Group • R = Role • U = User • Blank = Not applicable
OBJECTSCHEMA	VARCHAR (128)		Schema name of the object for which the audit policy is in use. OBJECTSCHEMA is null if OBJECTTYPE identifies an object to which a schema does not apply.
OBJECTNAME	VARCHAR (128)		Unqualified name of the object for which this audit policy is in use.

SYSCAT.COLAUTH

Each row represents a user, group, or role that has been granted one or more privileges on a column.

Table 40. SYSCAT.COLAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of a privilege.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of a privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
TABSCHEMA	VARCHAR (128)		Schema name of the table or view on which the privilege is held.
TABNAME	VARCHAR (128)		Unqualified name of the table or view on which the privilege is held.
COLNAME	VARCHAR (128)		Name of the column to which this privilege applies.
COLNO	SMALLINT		Column number of this column within the table (starting with 0).
PRIVTYPE	CHAR (1)		• R = Reference privilege • U = Update privilege

Table 40. SYSCAT.COLAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
GRANTABLE	CHAR (1)		• G = Privilege is grantable • N = Privilege is not grantable

Note:

1. Privileges can be granted by column, but can be revoked only on a table-wide basis.

SYSCAT.DBAUTH

Each row represents a user, group, or role that has been granted one or more database-level authorities.

Table 41. SYSCAT.DBAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the authority.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of the authority.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
BINDADDAUTH	CHAR (1)		Authority to create packages. • N = Not held • Y = Held
CONNECTAUTH	CHAR (1)		Authority to connect to the database. • N = Not held • Y = Held
CREATETABAUTH	CHAR (1)		Authority to create tables. • N = Not held • Y = Held
DBADMAUTH	CHAR (1)		DBADM authority. • N = Not held • Y = Held
EXTERNALROUTINEAUTH	CHAR (1)		Authority to create external routines. • N = Not held • Y = Held
IMPLSCHEMAAUTH	CHAR (1)		Authority to implicitly create schemas by creating objects in non-existent schemas. • N = Not held • Y = Held
LOADAUTH	CHAR (1)		Authority to use the DB2 load utility. • N = Not held • Y = Held

Table 41. SYSCAT.DBAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
NOFENCEAUTH	CHAR (1)		Authority to create non-fenced user-defined functions. • N = Not held • Y = Held
QUIESCECONNECTAUTH	CHAR (1)		Authority to access the database when it is quiesced. • N = Not held • Y = Held
LIBRARYADMAUTH	CHAR (1)		Reserved for future use.
SECURITYADMAUTH	CHAR (1)		Authority to administer database security. • N = Not held • Y = Held
SQLADMAUTH	CHAR (1)		Authority to monitor and tune SQL statements. • N = Not held • Y = Held
WLMADMAUTH	CHAR (1)		Authority to manage WLM objects. • N = Not held • Y = Held
EXPLAINAUTH	CHAR (1)		Authority to explain SQL statements without requiring actual privileges on the objects in the statement. • N = Not held • Y = Held
DATAACCESSAUTH	CHAR (1)		Authority to access data. • N = Not held • Y = Held
ACCESSCTRLAUTH	CHAR (1)		Authority to grant and revoke database object privileges. • N = Not held • Y = Held

SYSCAT.INDEXAUTH

Each row represents a user, group, or role that has been granted CONTROL privilege on an index.

Table 42. SYSCAT.INDEXAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the privilege.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.

Table 42. SYSCAT.INDEXAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
INDSCHEMA	VARCHAR (128)		Schema name of the index.
INDNAME	VARCHAR (128)		Unqualified name of the index.
CONTROLAUTH	CHAR (1)		CONTROL privilege. • N = Not held • Y = Held

SYSCAT.PACKAGEAUTH

Each row represents a user, group, or role that has been granted one or more privileges on a package.

Table 43. SYSCAT.PACKAGEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the privilege.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
PKGSCHEMA	VARCHAR (128)		Schema name of the package.
PKGNAME	VARCHAR (128)		Unqualified name of the package.
CONTROLAUTH	CHAR (1)		CONTROL privilege. • N = Not held • Y = Held
BINDAUTH	CHAR (1)		Privilege to bind the package. • G = Held and grantable • N = Not held • Y = Held
EXECUTEAUTH	CHAR (1)		Privilege to execute the package. • G = Held and grantable • N = Not held • Y = Held

SYSCAT.PACKAGEDEP

Each row represents a dependency of a package on some other object. The package depends on the object of type BTYPE of name BNAME, so a change to the object affects the package.

Table 44. SYSCAT.PACKAGEDEP Catalog View

Column Name	Data Type	Nullable	Description
PKGSCHEMA	VARCHAR (128)		Schema name of the package.
PKGNAME	VARCHAR (128)		Unqualified name of the package.
BINDER	VARCHAR (128)		Binder of the package.
BINDERTYPE	CHAR (1)		• U = Binder is an individual user
BTYPE	CHAR (1)		Type of object on which there is a dependency. Possible values are: • A = Table alias • B = Trigger • D = Server definition • F = Routine • G = User temporary table • I = Index • M = Function mapping • N = Nickname • O = Privilege dependency on all subtables or subviews in a table or view hierarchy • P = Page size • Q = Sequence object • R = User-defined data type • S = Materialized query table • T = Table (untyped) • U = Typed table • V = View (untyped) • W = Typed view • Z = XSR object • m = Module • n = Database partition group • q = Sequence alias • u = Module alias • v = Global variable
BSCHEMA	VARCHAR (128)		Schema name of an object on which the package depends.
BMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the object on which a dependency belongs. The null value if not a module object.
BNAME	VARCHAR (128)		Unqualified name of an object on which the package depends.
BMODULEID	INTEGER	Y	Identifier for the module of the object on which there is a dependency.
TABAUTH	SMALLINT	Y	If BTYPE is 'O', 'S', 'T', 'U', 'V', or 'W', encodes the privileges that are required by this package (SELECT, INSERT, UPDATE, or DELETE). Bit vector is defined in SQL.H.
VARAUTH	SMALLINT	Y	If BTYPE is 'v', encodes the privileges that are required by this package (READ or WRITE). Bit vector is defined in SQL.H.

Table 44. SYSCAT.PACKAGEDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
UNIQUE_ID	CHAR (8) FOR BIT DATA		Identifier for a specific package when multiple packages having the same name exist.
PKGVERSION	VARCHAR (64)	Y	Version identifier for the package.

Note:

1. If a function instance with dependencies is dropped, the package is put into an ″inoperative″ state, and it must be explicitly rebound. If any other object with dependencies is dropped, the package is put into an ″invalid″ state, and the system will attempt to rebind the package automatically when it is first referenced.

SYSCAT.PASSTHRUAUTH

Each row represents a user, group, or role that has been granted pass-through authorization to query a data source.

Table 45. SYSCAT.PASSTHRUAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the privilege.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
SERVERNAME	VARCHAR (128)		Name of the data source to which authorization is being granted.

SYSCAT.ROLEAUTH

Each row represents a role granted to a user, group, role, or PUBLIC.

Table 46. SYSCAT.ROLEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Authorization ID that granted the role.
GRANTORTYPE	CHAR (1)		• U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Authorization ID to which the role was granted.
GRANTEETYPE	CHAR (1)		• G = The grantee is a group • R = The grantee is a role • U = The grantee is an individual user
ROLENAME	VARCHAR (128)		Name of the role.
ROLEID	INTEGER		Identifier for the role.
ADMIN	CHAR (1)		Privilege to grant or revoke the role to or from others, or to comment on the role. • N = Not held • Y = Held

SYSCAT.ROLES

Each row represents a role.

Table 47. SYSCAT.ROLES Catalog View

Column Name	Data Type	Nullable	Description
ROLENAME	VARCHAR (128)		Name of the role.
ROLEID	INTEGER		Identifier for the role.
CREATE_TIME	TIMESTAMP		Time when the role was created.
AUDITPOLICYID	INTEGER	Y	Identifier for the audit policy.
AUDITPOLICYNAME	VARCHAR (128)	Y	Name of the audit policy.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.SCHEMAAUTH

Each row represents a user, group, or role that has been granted one or more privileges on a schema.

Table 48. SYSCAT.SCHEMAAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of a privilege.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of a privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
SCHEMANAME	VARCHAR (128)		Name of the schema to which this privilege applies.
ALTERINAUTH	CHAR (1)		Privilege to alter or comment on objects in the named schema. • G = Held and grantable • N = Not held • Y = Held
CREATEINAUTH	CHAR (1)		Privilege to create objects in the named schema. • G = Held and grantable • N = Not held • Y = Held
DROPINAUTH	CHAR (1)		Privilege to drop objects from the named schema. • G = Held and grantable • N = Not held • Y = Held

SYSCAT.ROUTINEAUTH

Each row represents a user, group, or role that has been granted EXECUTE privilege on:

- a particular routine (function, method, or procedure) in the database that is not defined in a module
- all routines in a particular schema in the database that are not defined in a module

Table 49. SYSCAT.ROUTINEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the privilege. 'SYSIBM' if the privilege was granted by the system.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
SCHEMA	VARCHAR (128)		Schema name of the routine.
SPECIFICNAME	VARCHAR (128)	Y	Specific name of the routine. If SPECIFICNAME is the null value and ROUTINETYPE is not 'M', the privilege applies to all routines of the type specified in ROUTINETYPE in the schema specified in SCHEMA. If SPECIFICNAME is the null value and ROUTINETYPE is 'M', the privilege applies to all methods for the subject type specified by TYPENAME in the schema specified by TYPESCHEMA. If SPECIFICNAME is the null value, ROUTINETYPE is 'M', and both TYPENAME and TYPESCHEMA are null values, the privilege applies to all methods for all types in the schema.
TYPESCHEMA	VARCHAR (128)	Y	Schema name of the type for the method. The null value if ROUTINETYPE is not 'M'.
TYPENAME	VARCHAR (128)	Y	Unqualified name of the type for the method. The null value if ROUTINETYPE is not 'M'. If TYPENAME is the null value and ROUTINETYPE is 'M', the privilege applies to all methods for any subject type if they are in the schema specified by SCHEMA.
ROUTINETYPE	CHAR (1)		Type of the routine. • F = Function • M = Method • P = Procedure
EXECUTEAUTH	CHAR (1)		Privilege to execute the routine. • G = Held and grantable • N = Not held • Y = Held

Table 49. SYSCAT.ROUTINEAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
GRANT_TIME	TIMESTAMP		Time at which the privilege was granted.

SYSCAT.SCHEMATA

Each row represents a schema.

Table 50. SYSCAT.SCHEMATA Catalog View

Column Name	Data Type	Nullable	Description
SCHEMANAME	VARCHAR (128)		Name of the schema.
OWNER	VARCHAR (128)		Authorization ID of the owner of the schema.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
DEFINER	VARCHAR (128)		Authorization ID of the definer of the schema or authorization ID of the owner of the schema if the ownership of the schema has been transferred.
DEFINERTYPE	CHAR (1)		• S = The definer is the system • U = The definer is an individual user
CREATE_TIME	TIMESTAMP		Time at which the schema was created.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.SEQUENCEAUTH

Each row represents a user, group, or role that has been granted one or more privileges on a sequence.

Table 51. SYSCAT.SEQUENCEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of a privilege.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of a privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
SEQSCHEMA	VARCHAR (128)		Schema name of the sequence.
SEQNAME	VARCHAR (128)		Unqualified name of the sequence.
ALTERAUTH	CHAR (1)		Privilege to alter the sequence. • G = Held and grantable • N = Not held • Y = Held

Table 51. SYSCAT.SEQUENCEAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
USAGEAUTH	CHAR (1)		Privilege to reference the sequence. • G = Held and grantable • N = Not held • Y = Held

SYSCAT.SECURITYLABELACCESS

Each row represents a security label that was granted to the database authorization ID.

Table 52. SYSCAT.SECURITYLABELACCESS Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the security label.
GRANTEE	VARCHAR (128)		Holder of the security label.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
SECLABELID	INTEGER		Identifier for the security label. For the name of the security label, select the SECLABELNAME column for the corresponding SECLABELID value in the SYSCAT.SECURITYLABELS catalog view.
SECPOLICYID	INTEGER		Identifier for the security policy that is associated with the security label. For the name of the security policy, select the SECPOLICYNAME column for the corresponding SECPOLICYID value in the SYSCAT.SECURITYPOLICIES catalog view.
ACCESSTYPE	CHAR (1)		• B = Both read and write access • R = Read access • W = Write access
GRANT_TIME	TIMESTAMP		Time at which the security label was granted.

SYSCAT.SECURITYLABELCOMPONENTELEMENTS

Each row represents an element value for a security label component.

Table 53. SYSCAT.SECURITYLABELCOMPONENTELEMENTS Catalog View

Column Name	Data Type	Nullable	Description
COMPID	INTEGER		Identifier for the security label component.
ELEMENTVALUE	VARCHAR (32)		Element value for the security label component.
ELEMENTVALUEENCODING	CHAR (8) FOR BIT DATA		Encoded form of the element value.

Table 53. SYSCAT.SECURITYLABELCOMPONENTELEMENTS Catalog View (continued)

Column Name	Data Type	Nullable	Description
PARENTELEMENTVALUE	VARCHAR (32)	Y	Name of the parent of an element for tree components; the null value for set and array components, and for the ROOT node of a tree component.

SYSCAT.SECURITYLABELCOMPONENTS

Each row represents a security label component.

Table 54. SYSCAT.SECURITYLABELCOMPONENTS Catalog View

Column Name	Data Type	Nullable	Description
COMPNAME	VARCHAR (128)		Name of the security label component.
COMPID	INTEGER		Identifier for the security label component.
COMPTYPE	CHAR (1)		Security label component type. • A = Array • S = Set • T = Tree
NUMELEMENTS	INTEGER		Number of elements in the security label component.
CREATE_TIME	TIMESTAMP		Time at which the security label component was created.
REMARKS	VARCHAR (254)		User-provided comments, or the null value.

SYSCAT.SECURITYLABELS

Each row represents a security label.

Table 55. SYSCAT.SECURITYLABELS Catalog View

Column Name	Data Type	Nullable	Description
SECLABELNAME	VARCHAR (128)		Name of the security label.
SECLABELID	INTEGER		Identifier for the security label.
SECPOLICYID	INTEGER		Identifier for the security policy to which the security label belongs.
SECLABEL	SYSPROC. DB2SECURITYLABEL		Internal representation of the security label.
CREATE_TIME	TIMESTAMP		Time at which the security label was created.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.SECURITYPOLICIES

Each row represents a security policy.

Table 56. SYSCAT.SECURITYPOLICIES Catalog View

Column Name	Data Type	Nullable	Description
SECPOLICYNAME	VARCHAR (128)		Name of the security policy.
SECPOLICYID	INTEGER		Identifier for the security policy.

Table 56. SYSCAT.SECURITYPOLICIES Catalog View (continued)

Column Name	Data Type	Nullable	Description
NUMSECLABELCOMP	INTEGER		Number of security label components in the security policy.
RWSECLABELREL	CHAR (1)		Relationship between the security labels for read and write access granted to the same authorization ID. • S = The security label for write access granted to a user is a subset of the security label for read access granted to that same user
NOTAUTHWRITESECLABEL	CHAR (1)		Action to take when a user is not authorized to write the security label that is specified in the INSERT or UPDATE statement. • O = Override • R = Restrict
CREATE_TIME	TIMESTAMP		Time at which the security policy was created.
GROUPAUTHS	CHAR (1)		Indicates if authorizations of security labels and exemptions granted to an authorization ID that represents a group will be used or ignored. • I = Ignored • U = Used
ROLEAUTHS	CHAR (1)		Indicates if authorizations of security labels and exemptions granted to an authorization ID that represents a role will be used or ignored. • I = Ignored • U = Used
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.SECURITYPOLICYCOMPONENTRULES

Each row represents the read and write access rules for a security label component of the security policy.

Table 57. SYSCAT.SECURITYPOLICYCOMPONENTRULES Catalog View

Column Name	Data Type	Nullable	Description
SECPOLICYID	INTEGER		Identifier for the security policy.
COMPID	INTEGER		Identifier for the security label component of the security policy.
ORDINAL	INTEGER		Position of the security label component as it appears in the security policy, starting with 1.
READACCESSRULENAME	VARCHAR (128)		Name of the read access rule that is associated with the security label component.
READACCESSRULETEXT	VARCHAR (512)		Text of the read access rule that is associated with the security label component.
WRITEACCESSRULENAME	VARCHAR (128)		Name of the write access rule that is associated with the security label component.

Table 57. SYSCAT.SECURITYPOLICYCOMPONENTRULES Catalog View (continued)

Column Name	Data Type	Nullable	Description
WRITEACCESSRULETEXT	VARCHAR (512)		Text of the write access rule that is associated with the security label component.

SYSCAT.SECURITYPOLICYEXEMPTIONS

Each row represents a security policy exemption that was granted to a database authorization ID.

Table 58. SYSCAT.SECURITYPOLICYEXEMPTIONS Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the exemption.
GRANTEE	VARCHAR (128)		Holder of the exemption.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
SECPOLICYID	INTEGER		Identifier for the security policy for which the exemption was granted. For the name of the security policy, select the SECPOLICYNAME column for the corresponding SECPOLICYID value in the SYSCAT.SECURITYPOLICIES catalog view.
ACCESSRULENAME	VARCHAR (128)		Name of the access rule for which the exemption was granted.
ACCESSTYPE	CHAR (1)		Type of access to which the rule applies. • R = Read access • W = Write access
ORDINAL	INTEGER		Position of the security label component in the security policy to which the rule applies.
ACTIONALLOWED	CHAR (1)		If the rule is DB2LBACWRITEARRAY, then: • D = Write down • U = Write up Blank otherwise.
GRANT_TIME	TIMESTAMP		Time at which the exemption was granted.

SYSCAT.SURROGATEAUTHIDS

Each row represents a user or a group that has been granted SETSESSIONUSER privilege on a user or PUBLIC.

Table 59. SYSCAT.SURROGATEAUTHIDS Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Authorization ID that granted TRUSTEDID the ability to act as a surrogate. When the TRUSTEDID represents a trusted context object, this field represents the authorization ID that created or altered the trusted context object.

Table 59. SYSCAT.SURROGATEAUTHIDS Catalog View (continued)

Column Name	Data Type	Nullable	Description
TRUSTEDID	VARCHAR (128)		Identifier for the entity that is trusted to act as a surrogate.
TRUSTEDIDTYPE	CHAR (1)		• C = Trusted context • G = Group • U = User
SURROGATEAUTHID	VARCHAR (128)		Surrogate authorization ID that can be assumed by TRUSTEDID. 'PUBLIC' indicates that TRUSTEDID can assume any authorization ID.
SURROGATEAUTHIDTYPE	CHAR (1)		• G = Group • U = User
AUTHENTICATE	CHAR (1)		• N = No authentication is required • Y = Authentication token is required with the authorization ID to authenticate the user before the authorization ID can be assumed • Blank = TRUSTEDIDTYPE is not 'C'
CONTEXTROLE	VARCHAR (128)	Y	A specific role to be assigned to the assumed authorization ID, which supercedes the default role, if any, that is defined for the trusted context. Null value when TRUSTEDIDTYPE is not 'C'.
GRANT_TIME	TIMESTAMP		Time at which the grant was made .

SYSCAT.SEQUENCES

Each row represents a sequence or alias.

Table 60. SYSCAT.SEQUENCES Catalog View

Column Name	Data Type	Nullable	Description
SEQSCHEMA	VARCHAR (128)		Schema name of the sequence.
SEQNAME	VARCHAR (128)		Unqualified name of the sequence.
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the sequence.
DEFINERTYPE	CHAR (1)		• S = The definer is the system • U = The definer is an individual user
OWNER	VARCHAR (128)		Authorization ID of the owner of the sequence.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
SEQID	INTEGER		Identifier for the sequence or alias.
SEQTYPE	CHAR (1)		Type of sequence. • A = Alias • I = Identity sequence • S = Sequence

Table 60. SYSCAT.SEQUENCES Catalog View (continued)

Column Name	Data Type	Nullable	Description
BASE_SEQSCHEMA	VARCHAR (128)	Y	If SEQTYPE is 'A', contains the schema name of the sequence or alias that is referenced by this alias; the null value otherwise.
BASE_SEQNAME	VARCHAR (128)	Y	If SEQTYPE is 'A', contains the unqualified name of the sequence or alias that is referenced by this alias; the null value otherwise.
INCREMENT	DECIMAL (31,0)	Y	Increment value. The null value if the sequence is an alias.
START	DECIMAL (31,0)	Y	Start value of the sequence. The null value if the sequence is an alias.
MAXVALUE	DECIMAL (31,0)	Y	Maximum value of the sequence. The null value if the sequence is an alias.
MINVALUE	DECIMAL (31,0)	Y	Minimum value of the sequence. The null value if the sequence is an alias.
NEXTCACHEFIRSTVALUE	DECIMAL (31,0)	Y	The first value available to be assigned in the next cache block. If no caching, the next value available to be assigned.
CYCLE	CHAR (1)		Indicates whether or not the sequence can continue to generate values after reaching its maximum or minimum value. • N = Sequence cannot cycle • Y = Sequence can cycle • Blank = Sequence is an alias.
CACHE	INTEGER		Number of sequence values to pre-allocate in memory for faster access. 0 indicates that values of the sequence are not to be preallocated. In a partitioned database, this value applies to each database partition. -1 if the sequence is an alias.
ORDER	CHAR (1)		Indicates whether or not the sequence numbers must be generated in order of request. • N = Sequence numbers are not required to be generated in order of request • Y = Sequence numbers must be generated in order of request • Blank = Sequence is an alias.
DATATYPEID	INTEGER		For built-in types, the internal identifier of the built-in type. For distinct types, the internal identifier of the distinct type. 0 if the sequence is an alias.
SOURCETYPEID	INTEGER		For a built-in type or if the sequence is an alias, this has a value of 0. For a distinct type, this is the internal identifier of the built-in type that is the source type for the distinct type.
CREATE_TIME	TIMESTAMP		Time at which the sequence was created.
ALTER_TIME	TIMESTAMP		Time at which the sequence was last altered.

Table 60. SYSCAT.SEQUENCES Catalog View (continued)

Column Name	Data Type	Nullable	Description
PRECISION	SMALLINT		Precision of the data type of the sequence. Possible values are: • 5 = SMALLINT • 10 = INTEGER • 19 = BIGINT For DECIMAL, it is the precision of the specified DECIMAL data type. 0 if the sequence is an alias.
ORIGIN	CHAR (1)		Origin of the sequence. • S = System-generated sequence • U = User-generated sequence
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.TABCONST

Each row represents a table constraint of type CHECK, UNIQUE, PRIMARY KEY, or FOREIGN KEY. For table hierarchies, each constraint is recorded only at the level of the hierarchy where the constraint was created.

Table 61. SYSCAT.TABCONST Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR (128)		Name of the constraint.
TABSCHEMA	VARCHAR (128)		Schema name of the table to which this constraint applies.
TABNAME	VARCHAR (128)		Unqualified name of the table to which this constraint applies.
OWNER	VARCHAR (128)		Authorization ID of the owner of the constraint.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
TYPE	CHAR (1)		Indicates the constraint type. • F = Foreign key • I = Functional dependency • K = Check • P = Primary key • U = Unique
ENFORCED	CHAR (1)		• N = Do not enforce constraint • Y = Enforce constraint
CHECKEXISTINGDATA	CHAR (1)		• D = Defer checking any existing data • I = Immediately check existing data • N = Never check existing data
ENABLEQUERYOPT	CHAR (1)		• N = Query optimization is disabled • Y = Query optimization is enabled

Table 61. SYSCAT.TABCONST Catalog View (continued)

Column Name	Data Type	Nullable	Description
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the constraint.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.TABLES

Each row represents a table, view, alias, or nickname. Each table or view hierarchy has one additional row representing the hierarchy table or hierarchy view that implements the hierarchy. Catalog tables and views are included.

Table 62. SYSCAT.TABLES Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the object.
TABNAME	VARCHAR (128)		Unqualified name of the object.
OWNER	VARCHAR (128)		Authorization ID of the owner of the table, view, alias, or nickname.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
TYPE	CHAR (1)		Type of object. • A = Alias • G = Created temporary table • H = Hierarchy table • L = Detached table • N = Nickname • S = Materialized query table • T = Table (untyped) • U = Typed table • V = View (untyped) • W = Typed view
STATUS	CHAR (1)		Status of the object. • C = Set integrity pending • N = Normal • X = Inoperative
BASE_TABSCHEMA	VARCHAR (128)	Y	If TYPE = 'A', contains the schema name of the table, view, alias, or nickname that is referenced by this alias; null value otherwise.
BASE_TABNAME	VARCHAR (128)	Y	If TYPE = 'A', contains the unqualified name of the table, view, alias, or nickname that is referenced by this alias; null value otherwise.
ROWTYPESCHEMA	VARCHAR (128)	Y	Schema name of the row type for this table, if applicable; null value otherwise.
ROWTYPENAME	VARCHAR (128)	Y	Unqualified name of the row type for this table, if applicable; null value otherwise.
CREATE_TIME	TIMESTAMP		Time at which the object was created.
ALTER_TIME	TIMESTAMP		Time at which the object was last altered.
INVALIDATE_TIME	TIMESTAMP		Time at which the object was last invalidated.

Table 62. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
STATS_TIME	TIMESTAMP	Y	Time at which any change was last made to recorded statistics for this object. The null value if statistics are not collected.
COLCOUNT	SMALLINT		Number of columns, including inherited columns (if any).
TABLEID	SMALLINT		Internal logical object identifier.
TBSPACEID	SMALLINT		Internal logical identifier for the primary table space for this object.
CARD	BIGINT		Total number of rows in the table; -1 if statistics are not collected.
NPAGES	BIGINT		Total number of pages on which the rows of the table exist; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
FPAGES	BIGINT		Total number of pages; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
OVERFLOW	BIGINT		Total number of overflow records in the table; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
TBSPACE	VARCHAR (128)	Y	Name of the primary table space for the table. If no other table space is specified, all parts of the table are stored in this table space. The null value for aliases, views, and partitioned tables.
INDEX_TBSPACE	VARCHAR (128)	Y	Name of the table space that holds all indexes created on this table. The null value for aliases, views, and partitioned tables, or if the INDEX IN clause was omitted or specified with the same value as the IN clause of the CREATE TABLE statement.
LONG_TBSPACE	VARCHAR (128)	Y	Name of the table space that holds all long data (LONG or LOB column types) for this table. The null value for aliases, views, and partitioned tables, or if the LONG IN clause was omitted or specified with the same value as the IN clause of the CREATE TABLE statement.
PARENTS	SMALLINT	Y	Number of parent tables for this object; that is, the number of referential constraints in which this object is a dependent.
CHILDREN	SMALLINT	Y	Number of dependent tables for this object; that is, the number of referential constraints in which this object is a parent.
SELFREFS	SMALLINT	Y	Number of self-referencing referential constraints for this object; that is, the number of referential constraints in which this object is both a parent and a dependent.
KEYCOLUMNS	SMALLINT	Y	Number of columns in the primary key.
KEYINDEXID	SMALLINT	Y	Index identifier for the primary key index; 0 or the null value if there is no primary key.
KEYUNIQUE	SMALLINT		Number of unique key constraints (other than the primary key constraint) defined on this object.
CHECKCOUNT	SMALLINT		Number of check constraints defined on this object.

Table 62. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
DATACAPTURE	CHAR (1)		• L = Table participates in data replication, including replication of LONG VARCHAR and LONG VARGRAPHIC columns • N = Table does not participate in data replication • Y = Table participates in data replication, excluding replication of LONG VARCHAR and LONG VARGRAPHIC columns
CONST_CHECKED	CHAR (32)		• Byte 1 represents foreign key constraint. • Byte 2 represents check constraint. • Byte 5 represents materialized query table. • Byte 6 represents generated column. • Byte 7 represents staging table. • Byte 8 represents data partitioning constraint. • Other bytes are reserved for future use. Possible values are: • F = In byte 5, the materialized query table cannot be refreshed incrementally. In byte 7, the content of the staging table is incomplete and cannot be used for incremental refresh of the associated materialized query table. • N = Not checked • U = Checked by user • W = Was in 'U' state when the table was placed in set integrity pending state • Y = Checked by system
PMAP_ID	SMALLINT	Y	Identifier for the distribution map that is currently in use by this table (the null value for aliases or views).
PARTITION_MODE	CHAR (1)		Indicates how data is distributed among database partitions in a partitioned database system. • H = Hashing • R = Replicated across database partitions • Blank = No database partitioning
LOG_ATTRIBUTE	CHAR (1)		• Always 0. This column is no longer used.
PCTFREE	SMALLINT		Percentage of each page to be reserved for future inserts.
APPEND_MODE	CHAR (1)		Controls how rows are inserted into pages. • N = New rows are inserted into existing spaces, if available • Y = New rows are appended to the end of the data
REFRESH	CHAR (1)		Refresh mode. • D = Deferred • I = Immediate • O = Once • Blank = Not a materialized query table
REFRESH_TIME	TIMESTAMP	Y	For REFRESH = 'D' or 'O', time at which the data was last refreshed (REFRESH TABLE statement); null value otherwise.

Table 62. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
LOCKSIZE	CHAR (1)		Indicates the preferred lock granularity for tables that are accessed by data manipulation language (DML) statements. Applies to tables only. Possible values are: • I = Block insert • R = Row • T = Table • Blank = Not applicable
VOLATILE	CHAR (1)		• C = Cardinality of the table is volatile • Blank = Not applicable
ROW_FORMAT	CHAR (1)		Not used.
PROPERTY	VARCHAR (32)		Properties for a table. A single blank indicates that the table has no properties. The following is position within string, value, and meaning: • 1, Y = User maintained materialized query table • 2, Y = Staging table • 3, Y = Propagate immediate • 11, Y = Nickname that will not be cached
STATISTICS_PROFILE	CLOB (10M)	Y	RUNSTATS command used to register a statistical profile for the object.
COMPRESSION	CHAR (1)		• B = Both value and row compression are activated • N = No compression is activated; a row format that does not support compression is used • R = Row compression is activated if licensed; a row format that supports compression might be used • V = Value compression is activated; a row format that supports compression is used • Blank = Not applicable
ACCESS_MODE	CHAR (1)		Access restriction state of the object. These states only apply to objects that are in set integrity pending state or to objects that were processed by a SET INTEGRITY statement. Possible values are: • D = No data movement • F = Full access • N = No access • R = Read-only access
CLUSTERED	CHAR (1)	Y	• Y = Table is multidimensionally clustered (even if only by one dimension) • Null value = Table is not multidimensionally clustered
ACTIVE_BLOCKS	BIGINT		Total number of active blocks in the table, or -1. Applies to multidimensional clustering (MDC) tables only.
DROPRULE	CHAR (1)		• N = No rule • R = Restrict rule applies on drop
MAXFREESPACESEARCH	SMALLINT		Reserved for future use.
AVGCOMPRESSEDROWSIZE	SMALLINT		Average length (in bytes) of compressed rows in this table; -1 if statistics are not collected.

Table 62. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
AVGROWCOMPRESSIONRATIO	REAL		For compressed rows in the table, this is the average compression ratio by row; that is, the average uncompressed row length divided by the average compressed row length; -1 if statistics are not collected.
AVGROWSIZE	SMALLINT		Average length (in bytes) of both compressed and uncompressed rows in this table; -1 if statistics are not collected.
PCTROWSCOMPRESSED	REAL		Compressed rows as a percentage of the total number of rows in the table; -1 if statistics are not collected.
LOGINDEXBUILD	VARCHAR (3)	Y	Level of logging that is to be performed during create, recreate, or reorganize index operations on the table. • OFF = Index build operations on the table will be logged minimally • ON = Index build operations on the table will be logged completely • Null value = Value of the *logindexbuild* database configuration parameter will be used to determine whether or not index build operations are to be completely logged
CODEPAGE	SMALLINT		Code page of the object. This is the default code page used for all character columns, triggers, check constraints, and expression-generated columns.
COLLATIONSCHEMA	VARCHAR (128)		Schema name of the collation for the table.
COLLATIONNAME	VARCHAR (128)		Unqualified name of the collation for the table.
COLLATIONSCHEMA_ORDERBY	VARCHAR (128)		Schema name of the collation for ORDER BY clauses in the table.
COLLATIONNAME_ORDERBY	VARCHAR (128)		Unqualified name of the collation for ORDER BY clauses in the table.
ENCODING_SCHEME	CHAR (1)		• A = CCSID ASCII was specified • U = CCSID UNICODE was specified • Blank = CCSID clause was not specified
PCTPAGESSAVED	SMALLINT		Approximate percentage of pages saved in the table as a result of row compression. This value includes overhead bytes for each user data row in the table, but does not include the space that is consumed by dictionary overhead; -1 if statistics are not collected.
LAST_REGEN_TIME	TIMESTAMP	Y	Time at which any views or check constraints on the table were last regenerated.
SECPOLICYID	INTEGER		Identifier for the security policy protecting the table; 0 for non-protected tables.
PROTECTIONGRANULARITY	CHAR (1)		• B = Both column- and row-level granularity • C = Column-level granularity • R = Row-level granularity • Blank = Non-protected table
AUDITPOLICYID	INTEGER	Y	Identifier for the audit policy.
AUDITPOLICYNAME	VARCHAR (128)	Y	Name of the audit policy.
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the table, view, alias, or nickname.

Table 62. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
ONCOMMIT	CHAR (1)		Specifies the action taken on the created temporary table when a COMMIT operation is performed. • D = Delete rows • P = Preserve rows • Blank = Table is not a created temporary table
LOGGED	CHAR (1)		Specifies whether the created temporary table is logged. • N = Not logged • Y = Logged • Blank = Table is not a created temporary table
ONROLLBACK	CHAR (1)		Specifies the action taken on the created temporary table when a ROLLBACK operation is performed. • D = Delete rows • P = Preserve rows • Blank = Table is not a created temporary table
LASTUSED	DATE		Reserved for future use.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.TABLESPACES

Each row represents a table space.

Table 63. SYSCAT.TABLESPACES Catalog View

Column Name	Data Type	Nullable	Description
TBSPACE	VARCHAR (128)		Name of the table space.
OWNER	VARCHAR (128)		Authorization ID of the owner of the table space.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
CREATE_TIME	TIMESTAMP		Time at which the table space was created.
TBSPACEID	INTEGER		Identifier for the table space.
TBSPACETYPE	CHAR (1)		Type of table space. • D = Database-managed space • S = System-managed space
DATATYPE	CHAR (1)		Type of data that can be stored in this table space. • A = All types of permanent data; regular table space • L = All types of permanent data; large table space • T = System temporary tables only • U = Created temporary tables or declared temporary tables only

Table 63. SYSCAT.TABLESPACES Catalog View (continued)

Column Name	Data Type	Nullable	Description
EXTENTSIZE	INTEGER		Size of each extent, in pages of size PAGESIZE. This many pages are written to one container in the table space before switching to the next container.
PREFETCHSIZE	INTEGER		Number of pages of size PAGESIZE to be read when prefetching is performed; -1 when AUTOMATIC.
OVERHEAD	DOUBLE		Controller overhead and disk seek and latency time, in milliseconds (average for the containers in this table space).
TRANSFERRATE	DOUBLE		Time to read one page of size PAGESIZE into the buffer (average for the containers in this table space).
WRITEOVERHEAD	DOUBLE	Y	Reserved for future use.
WRITETRANSFERRATE	DOUBLE	Y	Time to write one page of size PAGESIZE from the buffer to the table space (average for the containers in this table space). The null value means the same value as TRANSFERRATE will be used.
PAGESIZE	INTEGER		Size (in bytes) of pages in this table space.
DBPGNAME	VARCHAR (128)		Name of the database partition group that is associated with this table space.
BUFFERPOOLID	INTEGER		Identifier for the buffer pool that is used by this table space (1 indicates the default buffer pool).
DROP_RECOVERY	CHAR (1)		Indicates whether or not tables in this table space can be recovered after a drop table operation. • N = Tables are not recoverable • Y = Tables are recoverable
NGNAME[1]	VARCHAR (128)		Name of the database partition group that is associated with this table space.
DEFINER[2]	VARCHAR (128)		Authorization ID of the owner of the table space.
DATAPRIORITY	CHAR (1)		Reserved for future use.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The NGNAME column is included for backwards compatibility. See DBPGNAME.
2. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.TBSPACEAUTH

Each row represents a user, group, or role that has been granted the USE privilege on a particular table space in the database.

Table 64. SYSCAT.TBSPACEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the privilege.

Table 64. SYSCAT.TBSPACEAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
TBSPACE	VARCHAR (128)		Name of the table space.
USEAUTH	CHAR (1)		Privilege to create tables within the table space. • G = Held and grantable • N = Not held • Y = Held

SYSCAT.USEROPTIONS

Each row represents a server-specific user option value.

Table 65. SYSCAT.USEROPTIONS Catalog View

Column Name	Data Type	Nullable	Description
AUTHID	VARCHAR (128)		Local authorization ID, in uppercase characters.
AUTHIDTYPE	CHAR (1)		• U = Grantee is an individual user
SERVERNAME	VARCHAR (128)		Name of the server on which the user is defined.
OPTION	VARCHAR (128)		Name of the user option.
SETTING	VARCHAR (2048)		Value of the user option.

SYSCAT.VARIABLEAUTH

Each row represents a user, group, or role that has been granted one or more privileges by a specific grantor on a global variable in the database that is not defined in a module.

Table 66. SYSCAT.VARIABLEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the privilege.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user

Table 66. SYSCAT.VARIABLEAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
VARSCHEMA	VARCHAR (128)		Schema name of the global variable if VARMODULEID is null; otherwise schema name of the module to which the global variable belongs.
VARNAME	VARCHAR (128)		Unqualified name of the global variable.
VARID	INTEGER		Identifier for the global variable.
READAUTH	CHAR (1)		Privilege to read the global variable. • G = Held and grantable • N = Not held • Y = Held
WRITEAUTH	CHAR (1)		Privilege to write the global variable. • G = Held and grantable • N = Not held • Y = Held

SYSCAT.TABAUTH

Each row represents a user, group, or role that has been granted one or more privileges on a table or view.

Table 67. SYSCAT.TABAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the privilege.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
TABSCHEMA	VARCHAR (128)		Schema name of the table or view.
TABNAME	VARCHAR (128)		Unqualified name of the table or view.
CONTROLAUTH	CHAR (1)		CONTROL privilege. • N = Not held • Y = Held but not grantable
ALTERAUTH	CHAR (1)		Privilege to alter the table; allow a parent table to this table to drop its primary key or unique constraint; allow a table to become a materialized query table that references this table or view in the materialized query; or allow a table that references this table or view in its materialized query to no longer be a materialized query table. • G = Held and grantable • N = Not held • Y = Held

Table 67. SYSCAT.TABAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
DELETEAUTH	CHAR (1)		Privilege to delete rows from a table or updatable view. • G = Held and grantable • N = Not held • Y = Held
INDEXAUTH	CHAR (1)		Privilege to create an index on a table. • G = Held and grantable • N = Not held • Y = Held
INSERTAUTH	CHAR (1)		Privilege to insert rows into a table or updatable view, or to run the import utility against a table or view. • G = Held and grantable • N = Not held • Y = Held
REFAUTH	CHAR (1)		Privilege to create and drop a foreign key referencing a table as the parent. • G = Held and grantable • N = Not held • Y = Held
SELECTAUTH	CHAR (1)		Privilege to retrieve rows from a table or view, create views on a table, or to run the export utility against a table or view. • G = Held and grantable • N = Not held • Y = Held
UPDATEAUTH	CHAR (1)		Privilege to run the UPDATE statement against a table or updatable view. • G = Held and grantable • N = Not held • Y = Held

Chapter 26. Auditing database activities

Auditing DB2 database activities

DB2 database manager auditing activities are shown in this section.

Introduction to the DB2 audit facility

To manage access to your sensitive data, you can use a variety of authentication and access control mechanisms to establish rules and controls for acceptable data access. But to protect against and discover unknown or unacceptable behaviors you can monitor data access by using the DB2 audit facility.

Successful monitoring of unwanted data access and subsequent analysis can lead to improvements in the control of data access and the ultimate prevention of malicious or careless unauthorized access to data. The monitoring of application and individual user access, including system administration actions, can provide a historical record of activity on your database systems.

The DB2 audit facility generates, and allows you to maintain, an audit trail for a series of predefined database events. The records generated from this facility are kept in an audit log file. The analysis of these records can reveal usage patterns that would identify system misuse. Once identified, actions can be taken to reduce or eliminate such system misuse.

The audit facility provides the ability to audit at both the instance and the individual database level, independently recording all instance and database level activities with separate logs for each. The system administrator (who holds SYSADM authority) can use the db2audit tool to configure audit at the instance level as well as to control when such audit information is collected. The system administrator can use the db2audit tool to archive both instance and database audit logs as well as to extract audit data from archived logs of either type.

The security administrator (who holds SECADM authority within a database) can use audit policies in conjunction with the SQL statement, AUDIT, to configure and control the audit requirements for an individual database. The security administrator can use the following audit routines to perform the specified tasks:

- The SYSPROC.AUDIT_ARCHIVE stored procedure archives audit logs.
- The SYSPROC.AUDIT_LIST_LOGS table function allows you to locate logs of interest.
- The SYSPROC.AUDIT_DELIM_EXTRACT stored procedure extracts data into delimited files for analysis.

The security administrator can grant EXECUTE privilege on these routines to another user, therefore enabling the security administrator to delegate these tasks, if desired.

When working in a partitioned database environment, many of the auditable events occur at the database partition at which the user is connected (the coordinator partition) or at the catalog partition (if they are not the same database partition). The implication of this is that audit records can be generated by more

than one database partition. Part of each audit record contains information identifying the coordinator partition and originating partition (the partition where audit record originated).

At the instance level, the audit facility must be stopped and started explicitly by use of the db2audit start and db2audit stop commands. When you start instance-level auditing, the audit facility uses existing audit configuration information. Since the audit facility is independent of the DB2 database server, it will remain active even if the instance is stopped. In fact, when the instance is stopped, an audit record may be generated in the audit log. To start auditing at the database level, first you need to create an audit policy, then you associate this audit policy with the objects you want to monitor, such as, authorization IDs, database authorities, trusted contexts or particular tables.

Categories of audit records

There are different categories of audit records that may be generated. In the following description of the categories of events available for auditing, you should notice that following the name of each category is a one-word keyword used to identify the category type. The categories of events available for auditing are:

- Audit (AUDIT). Generates records when audit settings are changed or when the audit log is accessed.
- Authorization Checking (CHECKING). Generates records during authorization checking of attempts to access or manipulate DB2 database objects or functions.
- Object Maintenance (OBJMAINT). Generates records when creating or dropping data objects, and when altering certain objects.
- Security Maintenance (SECMAINT). Generates records when:
  - Granting or revoking object privileges or database authorities
  - Granting or revoking security labels or exemptions
  - Altering the group authorization, role authorization, or override or restrict attributes of an LBAC security policy
  - Granting or revoking the SETSESSIONUSER privilege
  - Modifying any of the SYSADM_GROUP, SYSCTRL_GROUP, SYSMAINT_GROUP, or SYSMON_GROUP configuration parameters.
- System Administration (SYSADMIN). Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.
- User Validation (VALIDATE). Generates records when authenticating users or retrieving system security information.
- Operation Context (CONTEXT). Generates records to show the operation context when a database operation is performed. This category allows for better interpretation of the audit log file. When used with the log's event correlator field, a group of events can be associated back to a single database operation. For example, a query statement for dynamic queries, a package identifier for static queries, or an indicator of the type of operation being performed, such as CONNECT, can provide needed context when analyzing audit results.

  Note: The SQL or XQuery statement providing the operation context might be very long and is completely shown within the CONTEXT record. This can make the CONTEXT record very large.
- Execute (EXECUTE). Generates records during the execution of SQL statements.

For any of the above categories, you can audit failures, successes, or both.

Any operations on the database server may generate several records. The actual number of records generated in the audit log depends on the number of categories of events to be recorded as specified by the audit facility configuration. It also depends on whether successes, failures, or both, are audited. For this reason, it is important to be selective of the events to audit.

Audit facility behavior

This topic provides background information to help you understand how the timing of writing audit records to the log can affect database performance; how to manage errors that occur within the audit facility; and how audit records are generated in different situations.

Controlling the timing of writing audit records to the active log

The writing of the audit records to the active log can take place synchronously or asynchronously with the occurrence of the events causing the generation of those records. The value of the *audit_buf_sz* database manager configuration parameter determines when the writing of audit records is done.

If the value of *audit_buf_sz* is zero (0), the writing is done synchronously. The event generating the audit record waits until the record is written to disk. The wait associated with each record causes the performance of the DB2 database to decrease.

If the value of *audit_buf_sz* is greater than zero, the record writing is done asynchronously. The value of the *audit_buf_sz* when it is greater than zero is the number of 4 KB pages used to create an internal buffer. The internal buffer is used to keep a number of audit records before writing a group of them out to disk. The statement generating the audit record as a result of an audit event will not wait until the record is written to disk, and can continue its operation.

In the asynchronous case, it could be possible for audit records to remain in an unfilled buffer for some time. To prevent this from happening for an extended period, the database manager forces the writing of the audit records regularly. An authorized user of the audit facility may also flush the audit buffer with an explicit request. Also, the buffers are automatically flushed during an archive operation.

There are differences when an error occurs dependent on whether there is synchronous or asynchronous record writing. In asynchronous mode there may be some records lost because the audit records are buffered before being written to disk. In synchronous mode there may be one record lost because the error could only prevent at most one audit record from being written.

Managing audit facility errors

The setting of the ERRORTYPE audit facility parameter controls how errors are managed between the DB2 database system and the audit facility. When the audit facility is active, and the setting of the ERRORTYPE audit facility parameter is `AUDIT`, then the audit facility is treated in the same way as any other part of DB2 database. An audit record must be written (to disk in synchronous mode; or to the audit buffer in asynchronous mode) for an audit event associated with a statement to be considered successful. Whenever an error is encountered when running in this mode, a negative SQLCODE is returned to the application for the statement generating an audit record.

If the error type is set to NORMAL, then any error from db2audit is ignored and the operation's SQLCODE is returned.

Audit records generated in different situations

Depending on the API or query statement and the audit settings, none, one, or several audit records may be generated for a particular event. For example, an SQL UPDATE statement with a SELECT subquery may result in one audit record containing the results of the authorization check for UPDATE privilege on a table and another record containing the results of the authorization check for SELECT privilege on a table.

For dynamic data manipulation language (DML) statements, audit records are generated for all authorization checking at the time that the statement is prepared. Reuse of those statements by the same user will not be audited again since no authorization checking takes place at that time. However, if a change has been made to one of the catalog tables containing privilege information, then in the next unit of work, the statement privileges for the cached dynamic SQL or XQuery statements are checked again and one or more new audit records created.

For a package containing only static DML statements, the only auditable event that could generate an audit record is the authorization check to see if a user has the privilege to execute that package. The authorization checking and possible audit record creation required for the static SQL or XQuery statements in the package is carried out at the time the package is precompiled or bound. The execution of the static SQL or XQuery statements within the package is auditable using the EXECUTE category. When a package is bound again either explicitly by the user, or implicitly by the system, audit records are generated for the authorization checks required by the static SQL or XQuery statements.

For statements where authorization checking is performed at statement execution time (for example, data definition language (DDL), GRANT, and REVOKE statements), audit records are generated whenever these statements are used.

Note: When executing DDL, the section number recorded for all events (except the context events) in the audit record will be zero (0) no matter what the actual section number of the statement might have been.

Working with DB2 audit data in DB2 tables

The following topics describe how to create DB2 audit data, how to create tables to hold this data, how to populate the tables with the DB2 audit data, and how to select the DB2 audit data from the tables.

Creating tables to hold the DB2 audit data

Before you can work with audit data in database tables, you need to create the tables to hold the data. You should consider creating these tables in a separate schema to isolate the data in the tables from unauthorized users.

Before you begin

- See the CREATE SCHEMA statement for the authorities and privileges that you require to create a schema.
- See the CREATE TABLE statement for the authorities and privileges that you require to create a table.
- Decide which table space you want to use to hold the tables. (This topic does not describe how to create table spaces.)

Note: The format of the tables you need to create to hold the audit data may change from release to release. New columns may be added or the size of an existing column may change. The script, `db2audit.ddl`, creates tables of the correct format to contain the audit records.

About this task

The examples that follow show how to create the tables to hold the records from the delimited files. If you want, you can create a separate schema to contain these tables.

If you do not want to use all of the data that is contained in the files, you can omit columns from the table definitions, or bypass creating certain tables, as required. If you omit columns from the table definitions, you must modify the commands that you use to load data into these tables.

1. Issue the db2 command to open a DB2 command window.
2. Optional. Create a schema to hold the tables. For this example, the schema is called AUDIT:

   ```
   CREATE SCHEMA AUDIT
   ```

3. Optional. If you created the AUDIT schema, switch to the schema before creating any tables:

   ```
   SET CURRENT SCHEMA = 'AUDIT'
   ```

4. Run the script, `db2audit.ddl`, to create the tables that will contain the audit records.

   The script `db2audit.ddl` is located in the `sqllib/misc` directory (`sqllib\misc` on Windows). The script assumes that a connection to the database exists and that an 8K table space is available. The command to run the script is: db2 +o -tf sqllib/misc/db2audit.ddl The tables that the script creates are: AUDIT, CHECKING, OBJMAINT, SECMAINT, SYSADMIN, VALIDATE, CONTEXT and EXECUTE.

5. After you have created the tables, the security administrator can use the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure, or the system administrator can use the db2audit extract command, to extract the audit records from the archived audit log files into delimited files. You can load the audit data from the delimited files into the database tables you just created.

Loading DB2 audit data into tables

After you have archived and extracted the audit log file into delimited files, and you have created the database tables to hold the audit data, you can load the audit data from the delimited files into the database tables for analysis.

About this task

You use the load utility to load the audit data into the tables. Issue a separate load command for each table. If you omitted one or more columns from the table definitions, you must modify the version of the LOAD command that you use to successfully load the data. Also, if you specified a delimiter character other than the default when you extracted the audit data, you must also modify the version of the LOAD command that you use.

1. Issue the db2 command to open a DB2 command window.
2. To load the AUDIT table, issue the following command:

   ```
   LOAD FROM audit.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
            INSERT INTO schema.AUDIT
   ```

Note: Specify the DELPRIORITYCHAR modifier to ensure proper parsing of binary data.

Note: Specify the LOBSINFILE option of the LOAD command (due to the restriction that any inline data for large objects must be limited to 32K). In some situations, you might also need to use the LOBS FROM option.

Note: When specifying the file name, use the fully qualified path name. For example, if you have the DB2 database system installed on the C: drive of a Windows-based computer, you would specify `C:\Program Files\IBM\SQLLIB\`*instance*`\security\audit.del` as the fully qualified file name for the `audit.del` file.

3. To load the CHECKING table, issue the following command:
   ```
   LOAD FROM checking.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
            INSERT INTO schema.CHECKING
   ```
4. To load the OBJMAINT table, issue the following command:
   ```
   LOAD FROM objmaint.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
            INSERT INTO schema.OBJMAINT
   ```
5. To load the SECMAINT table, issue the following command:
   ```
   LOAD FROM secmaint.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
            INSERT INTO schema.SECMAINT
   ```
6. To load the SYSADMIN table, issue the following command:
   ```
   LOAD FROM sysadmin.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
            INSERT INTO schema.SYSADMIN
   ```
7. To load the VALIDATE table, issue the following command:
   ```
   LOAD FROM validate.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
            INSERT INTO schema.VALIDATE
   ```
8. To load the CONTEXT table, issue the following command:
   ```
   LOAD FROM context.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
            INSERT INTO schema.CONTEXT
   ```
9. To load the EXECUTE table, issue the following command:
   ```
   LOAD FROM execute.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
            INSERT INTO schema.EXECUTE
   ```
10. After you finish loading the data into the tables, delete the `.del` files from the `security/auditdata` subdirectory of the `sqllib` directory.
11. When you have loaded the audit data into the tables, you are ready to select data from these tables for analysis.

If you have already populated the tables a first time, and want to do so again, use the INSERT option to have the new table data added to the existing table data. If you want to have the records from the previous db2audit extract operation removed from the tables, load the tables again using the REPLACE option.

Audit facility record layouts

When an audit record is extracted from the audit log, each record has one of the formats shown in the following tables. Each table is preceded by a sample record.

The description of each item in the record is shown one row at a time in the associated table. Each item is shown in the table in the same order as it is output in the delimited file after the extract operation.

Note:

1. Depending on the audit event, not all fields in the audit records will have values. When there is no values in the field, the field will not be shown in the audit output.
2. Some fields such as "Access Attempted" are stored in the delimited ASCII format as bit maps. In this flat report file, however, these fields appear as a set of strings representing the bit map values.

Details on audit facility record layouts

The various audit facility record layouts are shown in this section.

Audit record layout for AUDIT events

The following table shows the layout of the audit record for AUDIT events.

Sample audit record:

```
timestamp=2007-04-10-08.29.52.000001;
category=AUDIT;
audit event=START;
event correlator=0;
event status=0;
userid=newton;
authid=NEWTON;
application id=*LOCAL_APPLICATION;
application name=db2audit.exe;
```

Table 68. Audit Record Layout for AUDIT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: AUDIT
Audit Event	VARCHAR(32)	Specific Audit Event. For a list of possible values, refer to the section for the AUDIT category in "Audit events" on page 276.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.

Table 68. Audit Record Layout for AUDIT Events (continued)

NAME	FORMAT	DESCRIPTION
Package Section	SMALLINT	Section number in package being used at the time the audit event occurred
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.
Policy Name	VARCHAR(128)	The audit policy name.
Policy Association Object Type	CHAR(1)	The type of the object that the audit policy is associated with. Possible values include: • N = Nickname • S = MQT • T = Table (untyped) • i = Authorization ID • g= Authority • x = Trusted context • blank = Database
Policy Association Subobject Type	CHAR(1)	The type of sub-object that the audit policy is associated with. If the Object Type is ? (authorization id), then possible values are: • U = User • G = Group • R = Role
Policy Association Object Name	VARCHAR(128)	The name of the object that the audit policy is associated with.
Policy Association Object Schema	VARCHAR(128)	The schema name of the object that the audit policy is associated with. This is NULL if the Policy Association Object Type identifies an object to which a schema does not apply.

Table 68. Audit Record Layout for AUDIT Events (continued)

NAME	FORMAT	DESCRIPTION
Audit Status	CHAR(1)	The status of the AUDIT category in an audit policy. Possible values are: • B-Both • F-Failure • N-None • S-Success
Checking Status	CHAR(1)	The status of the CHECKING category in an audit policy. Possible values are: • B-Both • F-Failure • N-None • S-Success
Context Status	CHAR(1)	The status of the CONTEXT category in an audit policy. Possible values are: • B-Both • F-Failure • N-None • S-Success
Execute Status	CHAR(1)	The status of the EXECUTE category in an audit policy. Possible values are: • B-Both • F-Failure • N-None • S-Success
Execute With Data	CHAR(1)	The WITH DATA option of the EXECUTE category in the audit policy. Possible values are: • Y-WITH DATA • N-WITHOUT DATA
Objmaint Status	CHAR(1)	The status of the OBJMAINT category in an audit policy. Possible values are: • B-Both • F-Failure • N-None • S-Success
Secmaint Status	CHAR(1)	The status of the SECMAINT category in an audit policy. See Audit Status field for possible values.
Sysadmin Status	CHAR(1)	The status of the SYSADMIN category in an audit policy. Possible values are: • B-Both • F-Failure • N-None • S-Success

Table 68. Audit Record Layout for AUDIT Events (continued)

NAME	FORMAT	DESCRIPTION
Validate Status	CHAR(1)	The status of the VALIDATE category in an audit policy. Possible values are: • B-Both • F-Failure • N-None • S-Success
Error Type	CHAR(8)	The error type in an audit policy. Possible values are: AUDIT and NORMAL.
Data Path	VARCHAR(1024)	The path to the active audit logs specified on the db2audit configure command.
Archive Path	VARCHAR(1024)	The path to the archived audit logs specified on the db2audit configure command

Audit record layout for CHECKING events

The format of the audit record for CHECKING events is shown in the following table.

Sample audit record:

```
timestamp=1998-06-24-08.42.11.622984;
category=CHECKING;
audit event=CHECKING_OBJECT;
event correlator=2;
event status=0;
database=FOO;
userid=boss;
authid=BOSS;
application id=*LOCAL.newton.980624124210;
application name=testapp;
package schema=NULLID;
package name=SYSSH200;
package section=0;
object schema=GSTAGER;
object name=NONE;
object type=REOPT_VALUES;
access approval reason=DBADM;
access attempted=STORE;
```

Table 69. Audit record layout for CHECKING events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: CHECKING
Audit Event	VARCHAR(32)	Specific Audit Event. For a list of possible values, refer to the section for the CHECKING category in “Audit events” on page 276.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0

Table 69. Audit record layout for CHECKING events (continued)

NAME	FORMAT	DESCRIPTION
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Object Schema	VARCHAR(128)	Schema of the object for which the audit event was generated.
Object Name	VARCHAR(128)	Name of object for which the audit event was generated.
Object Type	VARCHAR(32)	Type of object for which the audit event was generated. Possible values include: those shown in the topic titled “Audit record object types”.
Access Approval Reason	CHAR(18)	Indicates the reason why access was approved for this audit event. Possible values include: those shown in the topic titled “List of possible CHECKING access approval reasons”.
Access Attempted	CHAR(18)	Indicates the type of access that was attempted. Possible values include: those shown in the topic titled “List of possible CHECKING access attempted types”.
Package Version	VARCHAR (64)	Version of the package in use at the time that the audit event occurred.
Checked Authorization ID	VARCHAR(128)	Authorization ID is checked when it is different than the authorization ID at the time of the audit event. For example, this can be the target owner in a TRANSFER OWNERSHIP statement. When the audit event is SWITCH_USER, this field represents the authorization ID that is switched to.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(128)	The name of the trusted context associated with the trusted connection.

Table 69. Audit record layout for CHECKING events (continued)

NAME	FORMAT	DESCRIPTION
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.

Audit record object types

The following table shows for each audit record object type whether it can generate CHECKING, OBJMAINT, and SECMAINT events.

Table 70. Audit Record Object Types Based on Audit Events

Object type	CHECKING events	OBJMAINT events	SECMAINT events
ACCESS_RULE			X
ALIAS	X	X	
ALL	X		
AUDIT_POLICY	X	X	
BUFFERPOOL	X	X	
CHECK_CONSTRAINT		X	
DATABASE	X		X
DATA TYPE		X	
EVENT_MONITOR	X	X	
FOREIGN_KEY		X	
FUNCTION	X	X	X
FUNCTION MAPPING	X	X	
GLOBAL_VARIABLE	X	X	X
HISTOGRAM TEMPLATE	X	X	
INDEX	X	X	X
INDEX EXTENSION		X	
INSTANCE	X		
JAR_FILE		X	
METHOD_BODY	X	X	X
MODULE	X	X	X
NICKNAME	X	X	X
NODEGROUP	X	X	
NONE	X	X	X
OPTIMIZATION PROFILE	X		
PACKAGE	X	X	X
PACKAGE CACHE	X		
PRIMARY_KEY		X	
REOPT_VALUES	X		
ROLE	X	X	X
SCHEMA	X	X	X
SECURITY LABEL		X	X

Table 70. Audit Record Object Types Based on Audit Events (continued)

Object type	CHECKING events	OBJMAINT events	SECMAINT events
SECURITY LABEL COMPONENT		X	
SECURITY POLICY		X	X
SEQUENCE	X	X	
SERVER	X	X	X
SERVER OPTION	X	X	
SERVICE CLASS	X	X	
STORED_PROCEDURE	X	X	X
SUMMARY TABLES	X	X	X
TABLE	X	X	X
TABLESPACE	X	X	X
THRESHOLD	X	X	
TRIGGER		X	
TRUSTED CONTEXT	X	X	X
TYPE MAPPING	X	X	
TYPE&TRANSFORM	X	X	
UNIQUE_CONSTRAINT		X	
USER MAPPING	X	X	
VIEW	X	X	X
WORK ACTION SET	X	X	
WORK CLASS SET	X	X	
WORKLOAD	X	X	X
WRAPPER	X	X	
XSR object	X	X	X

CHECKING access approval reasons

The following list shows the possible CHECKING access approval reasons.

Note that an audit record might contain multiple access approval reasons, for example: `access approval reason=DATAACCESS,ACCESSCTRL;`. When multiple access approval reasons are present, the user must have all stated authorities and privileges in order to pass the authorization check for the attempted access.

0x0000000000000001 ACCESS DENIED
Access is not approved; rather, it was denied.

0x0000000000000002 SYSADM
Access is approved; the application or user has SYSADM authority.

0x0000000000000004 SYSCTRL
Access is approved; the application or user has SYSCTRL authority.

0x0000000000000008 SYSMAINT
Access is approved; the application or user has SYSMAINT authority.

0x0000000000000010 DBADM
Access is approved; the application or user has DBADM authority.

0x0000000000000020 DATABASE
Access is approved; the application or user has an explicit privilege on the database.

0x0000000000000040 OBJECT
Access is approved; the application or user has a privilege on the object or function.

0x0000000000000080 DEFINER
Access is approved; the application or user is the definer of the object or function.

0x0000000000000100 OWNER
Access is approved; the application or user is the owner of the object or function.

0x0000000000000200 CONTROL
Access is approved; the application or user has CONTROL privilege on the object or function.

0x0000000000000400 BIND
Access is approved; the application or user has bind privilege on the package.

0x0000000000000800 SYSQUIESCE
Access is approved; if the instance or database is in quiesce mode, the application or user may connect or attach.

0x0000000000001000 SYSMON
Access is approved; the application or user has SYSMON authority.

0x0000000000002000 SECADM
Access is approved; the application or user has SECADM authority.

0x0000000000004000 SETSESSIONUSER
Access is approved; the application or user has SETSESSIONUSER authority.

0x0000000000008000 TRUSTED_CONTEXT_MATCH
Connection attributes matched the attributes of a unique trusted context defined at the DB2 server.

0x0000000000010000 TRUSTED_CONTEXT_USE
Access is approved to use a trusted context.

0x0000000000020000 SQLADM
Access is approved; the application or user has SQLADM authority.

0x0000000000040000 WLMADM
Access is approved; the application or user has WLMADM authority.

0x0000000000080000 EXPLAIN
Access is approved; the application or user has EXPLAIN authority.

0x0000000000100000 DATAACCESS
Access is approved; the application or user has DATAACCESS authority.

0x0000000000200000 ACCESSCTRL
Access is approved; the application or user has ACCESSSCTRL authority.

CHECKING access attempted types

The following list shows the possible CHECKING access attempted types.

If Audit Event is CHECKING_TRANSFER, then the audit entry reflects that a privilege is held or not.

0x0000000000000001 CONTROL
Attempt to verify if CONTROL privilege is held.

0x0000000000000002 ALTER
Attempt to alter an object or to verify if ALTER privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000004 DELETE
Attempt to delete an object or to verify if DELETE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000008 INDEX
Attempt to use an index or to verify if INDEX privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000010 INSERT
Attempt to insert into an object or to verify if INSERT privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000020 SELECT
Attempt to query a table or view or to verify if SELECT privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000040 UPDATE
Attempt to update data in an object or to verify if UPDATE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000080 REFERENCE
Attempt to establish referential constraints between objects or to verify if REFERENCE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000100 CREATE
Attempt to create an object.

0x0000000000000200 DROP
Attempt to drop an object.

0x0000000000000400 CREATEIN
Attempt to create an object within another schema.

0x0000000000000800 DROPIN
Attempt to drop an object found within another schema.

0x0000000000001000 ALTERIN
Attempt to alter or modify an object found within another schema.

0x0000000000002000 EXECUTE
Attempt to execute or run an application or to invoke a routine, create a function sourced from the routine (applies to functions only), or reference a routine in any DDL statement or to verify if EXECUTE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000004000 BIND
Attempt to bind or prepare an application.

0x0000000000008000 SET EVENT MONITOR
Attempt to set event monitor switches.

0x0000000000010000 SET CONSTRAINTS
Attempt to set constraints on an object.

0x0000000000020000 COMMENT ON
Attempt to create comments on an object.

0x0000000000040000 GRANT
Attempt to grant privileges or roles on an object to another authorization ID.

0x0000000000080000 REVOKE
Attempt to revoke privileges or roles from an object from an authorization ID.

0x0000000000100000 LOCK
Attempt to lock an object.

0x0000000000200000 RENAME
Attempt to rename an object.

0x0000000000400000 CONNECT
Attempt to connect to a database.

0x0000000000800000 Member of SYS Group
Attempt to access or use a member of the SYS group.

0x0000000001000000 Access All
Attempt to execute a statement with all required privileges on objects held (only used for DBADM/SYSADM).

0x0000000002000000 Drop All
Attempt to drop multiple objects.

0x0000000004000000 LOAD
Attempt to load a table in a table space.

0x0000000008000000 USE
Attempt to create a table in a table space or to verify if USE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000010000000 SET SESSION_USER
Attempt to execute the SET SESSION_USER statement.

0x0000000020000000 FLUSH
Attempt to execute the FLUSH statement.

0x0000000040000000 STORE
Attempt to view the values of a re-optimized statement in the EXPLAIN_PREDICATE table.

0x0000000400000000 TRANSFER
Attempt to transfer an object.

0x0000000800000000 ALTER_WITH_GRANT
Attempt to verify if ALTER with GRANT privilege is held.

0x0000001000000000 DELETE_WITH_GRANT
Attempt to verify if DELETE with GRANT privilege is held.

0x0000002000000000 INDEX_WITH_GRANT
Attempt to verify if INDEX with GRANT privilege is held

0x0000004000000000 INSERT_WITH_GRANT
Attempt to verify if INSERT with GRANT privilege is held.

0x0000008000000000 SELECT_WITH_GRANT
Attempt to verify if SELECT with GRANT privilege is held.

0x0000010000000000 UPDATE_WITH_GRANT
Attempt to verify if UPDATE with GRANT privilege is held.

0x0000020000000000 REFERENCE_WITH_GRANT
Attempt to verify if REFERENCE with GRANT privilege is held.

0x0000040000000000 USAGE
Attempt to use a sequence or an XSR object or to verify if USAGE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000080000000000 SET ROLE
Attempt to set a role.

0x0000100000000000 EXPLICIT_TRUSTED_CONNECTION
Attempt to establish an explicit trusted connection.

0x0000200000000000 IMPLICIT_TRUSTED_CONNECTION
Attempt to establish an implicit trusted connection.

0x0000400000000000 READ
Attempt to read a global variable.

0x0000800000000000 WRITE
Attempt to write a global variable.

0x0001000000000000 SWITCH_USER
Attempt to switch a user ID on an explicit trusted connection.

0x0002000000000000 AUDIT_USING
Attempt to associate an audit policy with an object.

0x0004000000000000 AUDIT_REPLACE
Attempt to replace an audit policy association with an object.

0x0008000000000000 AUDIT_REMOVE
Attempt to remove an audit policy association with an object.

0x0010000000000000 AUDIT_ARCHIVE
Attempt to archive the audit log.

0x0020000000000000 AUDIT_EXTRACT
Attempt to extract the audit log.

0x0040000000000000 AUDIT_LIST_LOGS
Attempt to list the audit logs.

0x0080000000000000 IGNORE_TRIGGERS
Attempt to ignore the triggers associated with a database object.

0x0100000000000000 PREPARE
Attempt to prepare an SQL statement and the user does not hold the necessary object level privilege or DATAACCESS authority.

0x0200000000000000 DESCRIBE
Attempt to describe a statement and the user does not hold the necessary object level privilege or DATAACCESS authority.

Audit record layout for OBJMAINT events

The format of the audit record for OBJMAINT events is shown in the following table.

Sample audit record:

```
timestamp=1998-06-24-08.42.41.957524;
category=OBJMAINT;
audit event=CREATE_OBJECT;
```

```
event correlator=3;
event status=0;
database=FOO;
userid=boss;
authid=BOSS;
application id=*LOCAL.newton.980624124210;
application name=testapp;
package schema=NULLID;
package name=SQLC28A1;
package section=0;
object schema=BOSS;
object name=AUDIT;
object type=TABLE;
```

Table 71. Audit Record Layout for OBJMAINT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: OBJMAINT
Audit Event	VARCHAR(32)	Specific Audit Event. For a list of possible values, refer to the section for the OBJMAINT category in “Audit events” on page 276.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(256)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Object Schema	VARCHAR(128)	Schema of the object for which the audit event was generated.
Object Name	VARCHAR(128)	Name of object for which the audit event was generated.
Object Type	VARCHAR(32)	Type of object for which the audit event was generated. Possible values include: those shown in the topic titled “Audit record object types”.
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Security Policy Name	VARCHAR(128)	The name of the security policy if the object type is TABLE and that table is associated with a security policy.

Table 71. Audit Record Layout for OBJMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Alter Action	VARCHAR(32)	Specific Alter operation Possible values include: • ADD_PROTECTED_COLUMN • ADD_COLUMN_PROTECTION • DROP_COLUMN_PROTECTION • ADD_ROW_PROTECTION • ADD_SECURITY_POLICY • ADD_ELEMENT • ADD COMPONENT • USE GROUP AUTHORIZATIONS • IGNORE GROUP AUTHORIZATIONS • USE ROLE AUTHORIZATIONS • IGNORE ROLE AUTHORIZATIONS • OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL • RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
Protected Column Name	VARCHAR(128)	If the Alter Action is ADD_COLUMN_PROTECTION or DROP_COLUMN_PROTECTION this is the name of the affected column.
Column Security Label	VARCHAR(128)	The security label protecting the column specified in the field Column Name.
Security Label Column Name	VARCHAR(128)	Name of the column containing the security label protecting the row.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.
Object Module	VARCHAR(128)	Name of module to which the object belongs.

Audit record layout for SECMAINT events

The format of the audit record for SECMAINT events is shown in the following table.

Sample audit record:

```
timestamp=1998-06-24-11.57.45.188101;
category=SECMAINT;
audit event=GRANT;
event correlator=4;
event status=0;
database=FOO;
userid=boss;
authid=BOSS;
application id=*LOCAL.boss.980624155728;
application name=db2bp;
package schema=NULLID;
package name=SQLC28A1;
package section=0;
object schema=BOSS;
object name=T1;
object type=TABLE;
grantor=BOSS;
grantee=WORKER;
grantee type=USER;
privilege=SELECT;
```

Table 72. Audit Record Layout for SECMAINT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: SECMAINT
Audit Event	VARCHAR(32)	Specific Audit Event. For a list of possible values, refer to the section for the SECMAINT category in “Audit events” on page 276.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.

Table 72. Audit Record Layout for SECMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Object Schema	VARCHAR(128)	Schema of the object for which the audit event was generated. If the object type field is ACCESS_RULE then this field contains the security policy name associated with the rule. The name of the rule is stored in the field Object Name. If the object type field is SECURITY_LABEL, then this field contains the name of the security policy that the security label is part of. The name of the security label is stored in the field Object Name.
Object Name	VARCHAR(128)	Name of object for which the audit event was generated. Represents a role name when the audit event is any of: • ADD_DEFAULT_ROLE • DROP_DEFAULT_ROLE • ALTER_DEFAULT_ROLE • ADD_USER • DROP_USER • ALTER_USER_ADD_ROLE • ALTER_USER_DROP_ROLE • ALTER_USER_AUTHENTICATION If the object type field is ACCESS_RULE then this field contains the name of the rule. The security policy name associated with the rule is stored in the field Object Schema. If the object type field is SECURITY_LABEL, then this field contains the name of the security label. The name of the security policy that it is part of is stored in the field Object Schema.
Object Type	VARCHAR(32)	Type of object for which the audit event was generated. Possible values include: those shown in the topic titled “Audit record object types”. The value is ROLE when the audit event is any of: • ADD_DEFAULT_ROLE • DROP_DEFAULT_ROLE • ALTER_DEFAULT_ROLE • ADD_USER • DROP_USER • ALTER_USER_ADD_ROLE • ALTER_USER_DROP_ROLE • ALTER_USER_AUTHENTICATION
Grantor	VARCHAR(128)	The ID of the grantor or the revoker of the privilege or authority.

Table 72. Audit Record Layout for SECMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Grantee	VARCHAR(128)	Grantee ID for which a privilege or authority was granted or revoked. Represents a trusted context object when the audit event is any of: • ADD_DEFAULT_ROLE • DROP_DEFAULT_ROLE • ALTER_DEFAULT_ROLE • ADD_USER, DROP_USER • ALTER_USER_ADD_ROLE • ALTER_USER_DROP_ROLE • ALTER_USER_AUTHENTICATION
Grantee Type	VARCHAR(32)	Type of the grantee that was granted to or revoked from. Possible values include: USER, GROUP, ROLE, AMBIGUOUS, or is TRUSTED_CONTEXT when the audit event is any of: • ADD_DEFAULT_ROLE • DROP_DEFAULT_ROLE • ALTER_DEFAULT_ROLE • ADD_USER • DROP_USER • ALTER_USER_ADD_ROLE • ALTER_USER_DROP_ROLE • ALTER_USER_AUTHENTICATION
Privilege or Authority	CHAR(34)	Indicates the type of privilege or authority granted or revoked. Possible values include: those shown in the topic titled “List of possible SECMAINT privileges or authorities”. The value is ROLE MEMBERSHIP when the audit event is any of the following: • ADD_DEFAULT_ROLE, DROP_DEFAULT_ROLE • ALTER_DEFAULT_ROLE • ADD_USER • DROP_USER • ALTER_USER_ADD_ROLE • ALTER_USER_DROP_ROLE • ALTER_USER_AUTHENTICATION
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.

Table 72. Audit Record Layout for SECMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Access Type	VARCHAR(32)	The access type for which a security label is granted. Possible values: • READ • WRITE • ALL The access type for which a security policy is altered. Possible values: • USE GROUP AUTHORIZATIONS • IGNORE GROUP AUTHORIZATIONS • USE ROLE AUTHORIZATIONS • IGNORE ROLE AUTHORIZATIONS • OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL • RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
Assumable Authid	VARCHAR(128)	When the privilege granted is a SETSESSIONUSER privilege this is the authorization ID that the grantee is allowed to set as the session user.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Grantor Type	VARCHAR(32)	Type of the grantor. Possible values include: USER.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context User	VARCHAR(128)	Identifies a trusted context user when the audit event is ADD_USER or DROP_USER.
Trusted Context User Authentication	INTEGER	Specifies the authentication setting for a trusted context user when the audit event is ADD_USER, DROP_USER or ALTER_USER_AUTHENTICATION 1 : Authentication is required 0 : Authentication is not required
Trusted Context Name	VARCHAR(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.

SECMAINT privileges or authorities

The following list shows the possible SECMAINT privileges or authorities.

0x00000000000000000000000000000001 Control Table
Control privilege granted or revoked on or from a table or view.

0x00000000000000000000000000000002 ALTER
Privilege granted or revoked to alter a table or sequence.

0x00000000000000000000000000000004 ALTER with GRANT
Privilege granted or revoked to alter a table or sequence with granting of privileges allowed.

0x00000000000000000000000000000008 DELETE TABLE
Privilege granted or revoked to drop a table or view.

0x00000000000000000000000000000010 DELETE TABLE with GRANT
Privilege granted or revoked to drop a table with granting of privileges allowed.

0x00000000000000000000000000000020 Table Index
Privilege granted or revoked on or from an index.

0x00000000000000000000000000000040 Table Index with GRANT
Privilege granted or revoked on or from an index with granting of privileges allowed.

0x00000000000000000000000000000080 Table INSERT
Privilege granted or revoked on or from an insert on a table or view.

0x00000000000000000000000000000100 Table INSERT with GRANT
Privilege granted or revoked on or from an insert on a table with granting of privileges allowed.

0x00000000000000000000000000000200 Table SELECT
Privilege granted or revoked on or from a select on a table.

0x00000000000000000000000000000400 Table SELECT with GRANT
Privilege granted or revoked on or from a select on a table with granting of privileges allowed.

0x00000000000000000000000000000800 Table UPDATE
Privilege granted or revoked on or from an update on a table or view.

0x00000000000000000000000000001000 Table UPDATE with GRANT
Privilege granted or revoked on or from an update on a table or view with granting of privileges allowed.

0x00000000000000000000000000002000 Table REFERENCE
Privilege granted or revoked on or from a reference on a table.

0x00000000000000000000000000004000 Table REFERENCE with GRANT
Privilege granted or revoked on or from a reference on a table with granting of privileges allowed.

0x00000000000000000000000000020000 CREATEIN Schema
CREATEIN privilege granted or revoked on or from a schema.

0x00000000000000000000000000040000 CREATEIN Schema with GRANT
CREATEIN privilege granted or revoked on or from a schema with granting of privileges allowed.

0x00000000000000000000000000080000 DROPIN Schema
DROPIN privilege granted or revoked on or from a schema.

0x00000000000000000000000000100000 DROPIN Schema with GRANT
DROPIN privilege granted or revoked on or from a schema with granting of privileges allowed.

0x00000000000000000000000000200000 ALTERIN Schema
ALTERIN privilege granted or revoked on or from a schema.

0x00000000000000000000000000400000 ALTERIN Schema with GRANT
ALTERIN privilege granted or revoked on or from a schema with granting of privileges allowed.

0x00000000000000000000000000800000 DBADM Authority
DBADM authority granted or revoked.

0x00000000000000000000000001000000 CREATETAB Authority
Createtab authority granted or revoked.

0x00000000000000000000000002000000 BINDADD Authority
Bindadd authority granted or revoked.

0x00000000000000000000000004000000 CONNECT Authority
CONNECT authority granted or revoked.

0x00000000000000000000000008000000 Create not fenced Authority
Create not fenced authority granted or revoked.

0x00000000000000000000000010000000 Implicit Schema Authority
Implicit schema authority granted or revoked.

0x00000000000000000000000020000000 Server PASSTHRU
Privilege granted or revoked to use the pass-through facility with this server (federated database data source).

0x00000000000000000000000040000000 ESTABLISH TRUSTED CONNECTION
Trusted connection was created

0x00000000000000000000000100000000 Table Space USE
Privilege granted or revoked to create a table in a table space.

0x00000000000000000000000200000000 Table Space USE with GRANT
Privilege granted or revoked to create a table in a table space with granting of privileges allowed.

0x00000000000000000000000400000000 Column UPDATE
Privilege granted or revoked on or from an update on one or more specific columns of a table.

0x00000000000000000000000800000000 Column UPDATE with GRANT
Privilege granted or revoked on or from an update on one or more specific columns of a table with granting of privileges allowed.

0x00000000000000000000001000000000 Column REFERENCE
Privilege granted or revoked on or from a reference on one or more specific columns of a table.

0x00000000000000000000002000000000 Column REFERENCE with GRANT
Privilege granted or revoked on or from a reference on one or more specific columns of a table with granting of privileges allowed.

0x00000000000000000000004000000000 LOAD Authority
LOAD authority granted or revoked.

0x00000000000000000000008000000000 Package BIND
BIND privilege granted or revoked on or from a package.

0x00000000000000000000010000000000 Package BIND with GRANT
BIND privilege granted or revoked on or from a package with granting of privileges allowed.

0x00000000000000000000020000000000 EXECUTE
EXECUTE privilege granted or revoked on or from a package or a routine.

0x00000000000000000000040000000000 EXECUTE with GRANT
EXECUTE privilege granted or revoked on or from a package or a routine with granting of privileges allowed.

0x00000000000000000000080000000000 EXECUTE IN SCHEMA
EXECUTE privilege granted or revoked for all routines in a schema.

0x00000000000000000000100000000000 EXECUTE IN SCHEMA with GRANT
EXECUTE privilege granted or revoked for all routines in a schema with granting of privileges allowed.

0x00000000000000000000200000000000 EXECUTE IN TYPE
EXECUTE privilege granted or revoked for all routines in a type.

0x00000000000000000000400000000000 EXECUTE IN TYPE with GRANT
EXECUTE privilege granted or revoked for all routines in a type with granting of privileges allowed.

0x00000000000000000000800000000000 CREATE EXTERNAL ROUTINE
CREATE EXTERNAL ROUTINE privilege granted or revoked.

0x00000000000000000001000000000000 QUIESCE_CONNECT
QUIESCE_CONNECT privilege granted or revoked.

0x00000000000000000004000000000000 SECADM Authority
SECADM authority granted or revoked

0x00000000000000000008000000000000 USAGE Authority
USAGE privilege granted or revoked on or from a sequence

0x00000000000000000010000000000000 USAGE with GRANT Authority
USAGE privilege granted or revoked on or from a sequence with granting of privileges allowed.

0x00000000000000000020000000000000 WITH ADMIN Option
WITH ADMIN Option is granted or revoked to or from a role.

0x00000000000000000040000000000000 SETSESSIONUSER Privilege
SETSESSIONUSER granted or revoked

0x00000000000000000080000000000000 Exemption
Exemption granted or revoked

0x00000000000000000100000000000000 Security label
Security label granted or revoked

0x00000000000000000200000000000000 WRITE with GRANT
Privilege granted or revoked to write a global variable with granting of privileges allowed.

0x00000000000000000400000000000000 Role Membership
Role membership that is granted or revoked

0x00000000000000000800000000000000 Role Membership with ADMIN Option
Role membership with ADMIN Option that is granted or revoked

0x00000000000000001000000000000000 READ
Privilege granted or revoked to read a global variable.

0x00000000000000002000000000000000 READ with GRANT
Privilege granted or revoked to read a global variable with granting of privileges allowed.

0x00000000000000004000000000000000 WRITE
Privilege granted or revoked to write a global variable.

0x00000000000000010000000000000000 SQLADM
SQLADM authority granted or revoked.

0x00000000000000020000000000000000 WLMADM
WLMADM authority granted or revoked.

0x00000000000000040000000000000000 EXPLAIN
EXPLAIN authority granted or revoked.

0x00000000000000080000000000000000 DATAACCESS
DATAACCESS authority granted or revoked.

0x00000000000000100000000000000000 ACCESSCTRL
ACCESSCTRL authority granted or revoked.

Audit record layout for SYSADMIN events

The following table shows the audit record layout for SYSADMIN events.

Sample audit record:

```
timestamp=1998-06-24-11.54.04.129923;
category=SYSADMIN;
audit event=DB2AUDIT;
event correlator=1;
event status=0;
userid=boss;authid=BOSS;
application id=*LOCAL.boss.980624155404;
application name=db2audit;
```

Table 73. Audit Record Layout for SYSADMIN Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: SYSADMIN
Audit Event	VARCHAR(32)	Specific Audit Event. For a list of possible values, refer to the section for the SYSADMIN category in “Audit events” on page 276.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.

Table 73. Audit Record Layout for SYSADMIN Events (continued)

NAME	FORMAT	DESCRIPTION
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.

Audit events

For each audit category, certain types of events can create audit records.

Events for the AUDIT category

- ALTER_AUDIT_POLICY
- ARCHIVE
- AUDIT_REMOVE
- AUDIT_REPLACE
- AUDIT_USING
- CONFIGURE
- CREATE_AUDIT_POLICY
- DB2AUD
- DROP_AUDIT_POLICY
- EXTRACT
- FLUSH
- LIST_LOGS
- PRUNE (not generated in Version 9.5, and later).
- START
- STOP
- UPDATE_DBM_CFG

Events for the CHECKING category

- CHECKING_FUNCTION
- CHECKING_MEMBERSHIP_IN_ROLES
- CHECKING_OBJECT
- CHECKING_TRANSFER

Events for the CONTEXT category

Table 74. Events for the CONTEXT category

CONNECT	SET_APPL_PRIORITY
CONNECT_RESET	RESET_DB_CFG
ATTACH	GET_DB_CFG
DETACH	GET_DFLT_CFG
DARI_START	UPDATE_DBM_CFG
DARI_STOP	SET_MONITOR
BACKUP_DB	GET_SNAPSHOT
RESTORE_DB	ESTIMATE_SNAPSHOT_SIZE
ROLLFORWARD_DB	RESET_MONITOR
OPEN_TABLESPACE_QUERY	OPEN_HISTORY_FILE
FETCH_TABLESPACE	CLOSE_HISTORY_FILE
CLOSE_TABLESPACE_QUERY	FETCH_HISTORY_FILE
OPEN_CONTAINER_QUERY	SET_RUNTIME_DEGREE
CLOSE_CONTAINER_QUERY	UPDATE_AUDIT
FETCH_CONTAINER_QUERY	DBM_CFG_OPERATION
SET_TABLESPACE_CONTAINERS	DISCOVER
GET_TABLESPACE_STATISTIC	OPEN_CURSOR
READ_ASYNC_LOG_RECORD	CLOSE_CURSOR
QUIESCE_TABLESPACE	FETCH_CURSOR
LOAD_TABLE	EXECUTE
UNLOAD_TABLE	EXECUTE_IMMEDIATE
UPDATE_RECOVERY_HISTORY	PREPARE
PRUNE_RECOVERY_HISTORY	DESCRIBE
SINGLE_TABLESPACE_QUERY	BIND
LOAD_MSG_FILE	REBIND
UNQUIESCE_TABLESPACE	RUNSTATS
ENABLE_MULTIPAGE	REORG
DESCRIBE_DATABASE	REDISTRIBUTE
DROP_DATABASE	COMMIT
CREATE_DATABASE	ROLLBACK
ADD_NODE	REQUEST_ROLLBACK
FORCE_APPLICATION	IMPLICIT_REBIND
	EXTERNAL_CANCEL
	SWITCH_USER

Events for the EXECUTE category

- COMMIT Execution of a COMMIT statement
- CONNECT Establishment of a database connection
- CONNECT RESET Termination of a database connection
- DATA A host variable or parameter marker data values for the statement

  This event is repeated for each host variable or parameter marker that is part of the statement. It is only present in a delimited extract of an audit log.
- GLOBAL COMMIT Execution of a COMMIT within a global transaction
- GLOBAL ROLLBACK Execution of a ROLLBACK within a global transaction
- RELEASE SAVEPOINT Execution of a RELEASE SAVEPOINT statement

- ROLLBACK Execution of a ROLLBACK statement
- SAVEPOINT Execution of a SAVEPOINT statement
- STATEMENT Execution of an SQL statement
- SWITCH USER Switching of a user within a trusted connection

Events for the OBJMAINT category

- ALTER_OBJECT (generated when altering protected tables and when altering modules)
- CREATE_OBJECT
- DROP_OBJECT
- RENAME_OBJECT

Events for the SECMAINT category

- ADD_DEFAULT_ROLE
- ADD_USER
- ALTER_DEFAULT_ROLE
- ALTER SECURITY POLICY
- ALTER_USER_ADD_ROLE
- ALTER_USER_AUTHENTICATION
- ALTER_USER_DROP_ROLE
- DROP_DEFAULT_ROLE
- DROP_USER
- GRANT
- IMPLICIT_GRANT
- IMPLICIT_REVOKE
- REVOKE
- SET_SESSION_USER
- TRANSFER_OWNERSHIP
- UPDATE_DBM_CFG

Events for the SYSADMIN category

Table 75. Events for the SYSADMIN category

START_DB2	ROLLFORWARD_DB
STOP_DB2	SET_RUNTIME_DEGREE
CREATE_DATABASE	SET_TABLESPACE_CONTAINERS
ALTER_DATABASE	UNCATALOG_DB
DROP_DATABASE	UNCATALOG_DCS_DB
UPDATE_DBM_CFG	UNCATALOG_NODE
UPDATE_DB_CFG	UPDATE_ADMIN_CFG
CREATE_TABLESPACE	UPDATE_MON_SWITCHES
DROP_TABLESPACE	LOAD_TABLE
ALTER_TABLESPACE	DB2AUDIT
RENAME_TABLESPACE	SET_APPL_PRIORITY
CREATE_NODEGROUP	CREATE_DB_AT_NODE
DROP_NODEGROUP	KILLDBM
ALTER_NODEGROUP	MIGRATE_SYSTEM_DIRECTORY
CREATE_BUFFERPOOL	DB2REMOT
DROP_BUFFERPOOL	DB2AUD
ALTER_BUFFERPOOL	MERGE_DBM_CONFIG_FILE
CREATE_EVENT_MONITOR	UPDATE_CLI_CONFIGURATION
DROP_EVENT_MONITOR	OPEN_TABLESPACE_QUERY
ENABLE_MULTIPAGE	SINGLE_TABLESPACE_QUERY
MIGRATE_DB_DIR	CLOSE_TABLESPACE_QUERY
DB2TRC	FETCH_TABLESPACE
DB2SET	OPEN_CONTAINER_QUERY
ACTIVATE_DB	FETCH_CONTAINER_QUERY
ADD_NODE	CLOSE_CONTAINER_QUERY
BACKUP_DB	GET_TABLESPACE_STATISTICS
CATALOG_NODE	DESCRIBE_DATABASE
CATALOG_DB	ESTIMATE_SNAPSHOT_SIZE
CATALOG_DCS_DB	READ_ASYNC_LOG_RECORD
CHANGE_DB_COMMENT	PRUNE_RECOVERY_HISTORY
DEACTIVATE_DB	UPDATE_RECOVERY_HISTORY
DROP_NODE_VERIFY	QUIESCE_TABLESPACE
FORCE_APPLICATION	UNLOAD_TABLE
GET_SNAPSHOT	UPDATE_DATABASE_VERSION
LIST_DRDA_INDOUBT_TRANSACTIONS	CREATE_INSTANCE
MIGRATE_DB	DELETE_INSTANCE
RESET_ADMIN_CFG	SET_EVENT_MONITOR
RESET_DB_CFG	GRANT_DBADM (V97:no longer generated)
RESET_DBM_CFG	REVOKE_DBADM (V97:no longer generated)
RESET_MONITOR	GRANT_DB_AUTH (V97:no longer generated)
RESTORE_DB	REVOKE_DB_AUTH (V97:no longer generated)
	REDISTRIBUTE_NODEGROUP

Events for the VALIDATE category

- AUTHENTICATE
- CHECK_GROUP_MEMBERSHIP (not generated in Version 9.5, and later)
- GET_USERMAPPING_FROM_PLUGIN
- GET_GROUPS (not generated in Version 9.5, and later)
- GET_USERID (not generated in Version 9.5, and later)

Audit record layout for VALIDATE events

The format of the audit record for VALIDATE events is shown in the following table.

Sample audit record:

```
timestamp=2007-05-07-10.30.51.585626;
category=VALIDATE;
audit event=AUTHENTICATION;
event correlator=1;
event status=0;
userid=newton;
authid=NEWTON;
execution id=gstager;
application id=*LOCAL.gstager.070507143051;
application name=db2bp;
auth type=SERVER;
plugin name=IBMOSauthserver;
```

Table 76. Audit Record Layout for VALIDATE Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: VALIDATE
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: GET_GROUPS, GET_USERID, AUTHENTICATE_PASSWORD, VALIDATE_USER, AUTHENTICATION and GET_USERMAPPING_FROM_PLUGIN.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Execution ID	VARCHAR(1024)	Execution ID in use at the time of the audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Authentication Type	VARCHAR(32)	Authentication type at the time of the audit event.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Plug-in Name	VARCHAR(32)	The name of the plug-in in use at the time the audit event occurred.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.

Table 76. Audit Record Layout for VALIDATE Events (continued)

NAME	FORMAT	DESCRIPTION
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The name of the role inherited through the trusted context.

Audit record layout for CONTEXT events

The following table shows the audit record layout for CONTEXT events.

Sample audit record:

```
timestamp=1998-06-24-08.42.41.476840;
category=CONTEXT;
audit event=EXECUTE_IMMEDIATE;
event correlator=3;
database=FOO;
userid=boss;
authid=BOSS;
application id=*LOCAL.newton.980624124210;
application name=testapp;
package schema=NULLID;
package name=SQLC28A1;
package section=203;
text=create table audit(c1 char(10), c2 integer);
```

Table 77. Audit Record Layout for CONTEXT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: CONTEXT
Audit Event	VARCHAR(32)	Specific Audit Event. For a list of possible values, refer to the section for the CONTEXT category in “Audit events” on page 276.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.

Table 77. Audit Record Layout for CONTEXT Events (continued)

NAME	FORMAT	DESCRIPTION
User ID	VARCHAR(1024)	User ID at time of audit event. When the audit event is SWITCH_USER, this field represents the user ID that is switched to.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event. When the audit event is SWITCH_USER, this field represents the authorization ID that is switched to.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Statement Text	CLOB(8M)	Text of the SQL or XQuery statement, if applicable. Null if no SQL or XQuery statement text is available.
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.

Audit facility tips and techniques

Best practices for managing your audit include regularly archiving the audit log, using the error type AUDIT when you create an audit policy, and other tips as described here.

Archiving the audit log

You should archive the audit log on a regular basis. Archiving the audit log moves the current audit log to an archive directory while the server begins writing to a new, active audit log. The name of each archived log file includes a timestamp that helps you identify log files of interest for later analysis.

For long term storage, you may want to compress groups of archived files.

For archived audit logs that you are no longer interested in, the instance owner can simply delete the files from the operating system.

Error handling

When you create an audit policy, you should use the error type AUDIT, unless you are just creating a test audit policy. For example, if the error type is set to AUDIT, and an error occurs, such as running out of disk space, then an error is returned. The error condition must be corrected before any more auditable actions can continue. However, if the error type had been set to NORMAL, the logging would simply fail and no error is returned to the user. Operation continues as if the error did not happen.

If a problem occurs during archive, such as running out of disk space in the archive path, or the archive path does not exist, the archive process fails and an interim log file with the file extension `.bk` is generated in the audit log data path, for example, `db2audit.instance.log.0.20070508172043640941.bk`. After the problem is resolved (by allocating sufficient disk space in the archive path, or by creating the archive path) you must move this interim log to the archive path. Then, you can treat it in the same way as a successfully archived log.

DDL statement restrictions

Some data definition language (DDL) statements, called AUDIT exclusive SQL statements, do not take effect until the next unit of work. Therefore, you are advised to use a COMMIT statement immediately after each of these statements.

The AUDIT exclusive SQL statements are:

- AUDIT
- CREATE AUDIT POLICY, ALTER AUDIT POLICY, and DROP AUDIT POLICY
- DROP ROLE and DROP TRUSTED CONTEXT, if the role or trusted context being dropped is associated with an audit policy

Table format for holding archived data may change

The security administrator can use the SYSPROC.AUDIT_DEL_EXTRACT stored procedure, or the system administrator can use the db2audit extract command, to extract audit records from the archived audit log files into delimited files. You can load the audit data from the delimited files into DB2 database tables for analysis. The format of the tables you need to create to hold the audit data may change from release to release.

Important: The script, `db2audit.ddl`, creates tables of the correct format to contain the audit records. You should expect to run `db2audit.ddl` for each release, as columns may be added or the size of an existing column may change.

Using CHECKING events

In most cases, when working with CHECKING events, the object type field in the audit record is the object being checked to see if the required privilege or authority is held by the user ID attempting to access the object. For example, if a user attempts to ALTER a table by adding a column, then the CHECKING event audit record indicates the access attempted was "ALTER" and the object type being checked was "TABLE" (not the column, because it is table privileges that are checked).

However, when the checking involves verifying if a database authority exists to allow a user ID to CREATE or BIND an object, or to DROP an object, then although there is a check against the database, the object type field will specify the object being created, bound, or dropped (rather than the database itself).

When creating an index on a table, the privilege to create an index is required, therefore the CHECKING event audit record has an access attempt type of "index" rather than "create".

Audit records created for binding a package

When binding a package that already exists, then an OBJMAINT event audit record is created for the DROP of the package and then another OBJMAINT event audit record is created for the CREATE of the new copy of the package.

Using CONTEXT event information after ROLLBACK

Data Definition Language (DDL) may generate OBJMAINT or SECMAINT events that are logged as successful. It is possible however that following the logging of the event, a subsequent error may cause a ROLLBACK to occur. This would leave the object as not created; or the GRANT or REVOKE actions as incomplete. The use of CONTEXT events becomes important in this case. Such CONTEXT event audit records, especially the statement that ends the event, indicates the nature of the completion of the attempted operation.

The load delimiter

When extracting audit records in a delimited format suitable for loading into a DB2 database table, you should be clear regarding the delimiter used within the statement text field. This can be done when extracting the delimited file, using:

```
db2audit extract delasc delimiter <load delimiter>
```

The *load delimiter* can be a single character (such as ″) or a four-byte string representing a hexadecimal value (such as "0xff"). Examples of valid commands are:

```
db2audit extract delasc
db2audit extract delasc delimiter !
db2audit extract delasc delimiter 0xff
```

If you have used anything other than the default load delimiter as the delimiter when extracting, you should use the MODIFIED BY option on the LOAD command. A partial example of the LOAD command with "0xff" used as the delimiter follows:

```
db2 load from context.del of del modified by chardel0xff replace into ...
```

This will override the default load character string delimiter which is " (double quote).

Audit policies

The security administrator can use audit policies to configure the audit facility to gather information only about the data and objects that are needed.

The security administrator can create audit policies to control what is audited within an individual database. The following objects can have an audit policy associated with them:

- The whole database

  All auditable events that occur within the database are audited according to the audit policy.
- Tables

  All data manipulation language (DML) and XQUERY access to the table (untyped), MQT (materialized query table), or nickname is audited. Only EXECUTE category audit events with or without data are generated when the table is accessed even if the policy indicates that other categories should be audited.
- Trusted contexts

  All auditable events that happen within a trusted connection defined by the particular trusted context are audited according to the audit policy.
- Authorization IDs representing users, groups, or roles

  All auditable events that are initiated by the specified user are audited according to the audit policy.

  All auditable events that are initiated by users that are a member of the group or role are audited according to the audit policy. Indirect role membership, such as through other roles or groups, is also included.

  You can capture similar data by using the Work Load Management event monitors by defining a work load for a group and capturing the activity details. You should be aware that the mapping to workloads can involve attributes in addition to just the authorization ID, which can cause you to not achieve the desired granularity in auditing, or if those other attributes are modified, connections may map to different (possibly unmonitored) workloads. The auditing solution provides a guarantee that a user, group or role will be audited.
- Authorities (SYSADM, SECADM, DBADM, SQLADM, WLMADM, ACCESSCTRL, DATAACCESS, SYSCTRL, SYSMAINT, SYSMON)

  All auditable events that are initiated by a user that holds the specified authority, even if that authority is unnecessary for the event, are audited according to the audit policy.

The security administrator can create multiple audit policies. For example, your company might want a policy for auditing sensitive data and a policy for auditing the activity of users holding DBADM authority. If multiple audit policies are in effect for a statement, all events required to be audited by each of the audit policies are audited (but audited only once). For example, if the database's audit policy requires auditing successful EXECUTE events for a particular table and the user's audit policy requires auditing failures of EXECUTE events for that same table, both successful and failed attempts at accessing that table are audited.

For a specific object, there can only be one audit policy in effect. For example, you cannot have multiple audit policies associated with the same table at the same time.

An audit policy cannot be associated with a view or a typed table. Views that access a table that has an associated audit policy are audited according to the underlying table's policy.

The audit policy that applies to a table does not automatically apply to a MQT based on that table. If you associate an audit policy with a table, associate the same policy with any MQT based on that table.

Auditing performed during a transaction is done based on the audit policies and their associations at the start of the transaction. For example, if the security administrator associates an audit policy with a user and that user is in a transaction at the time, the audit policy does not affect any remaining statements performed within that transaction. Also, changes to an audit policy do not take effect until they are committed. If the security administrator issues an ALTER AUDIT POLICY statement, it does not take effect until the statement is committed.

The security administrator uses the CREATE AUDIT POLICY statement to create an audit policy, and the ALTER AUDIT POLICY statement to modify an audit policy. These statements can specify:

- The status values for events to be audited: None, Success, Failure, or Both.
  Only auditable events that match the specified status value are audited.
- The server behavior when errors occur during auditing.

The security administrator uses the AUDIT statement to associate an audit policy with the current database or with a database object, at the current server. Any time the object is in use, it is audited according to this audit policy.

To delete an audit policy, the security administrator uses the DROP statement. You cannot drop an audit policy if it is associated with any object. Use the AUDIT REMOVE statement to remove any remaining association with an object. To add metadata to an audit policy, the security administrator uses the COMMENT statement.

Events generated before a full connection has been established

For some events generated during connect and a switch user operation, the only audit policy information available is the policy that is associated with the database. These events are shown in the following table:

Table 78. Connection events

Event	Audit category	Comment
CONNECT	CONTEXT	
CONNECT_RESET	CONTEXT	
AUTHENTICATION	VALIDATE	This includes authentication during both connect and switch user within a trusted connection.
CHECKING_FUNC	CHECKING	The access attempted is SWITCH_USER.

These events are audited based only on the audit policy associated with the database and not with audit policies associated with any other object such as a user, their groups, or authorities. For the CONNECT and AUTHENTICATION events that occur during connect, the instance-level audit settings are used until the database is activated. The database is activated either during the first connection or when the ACTIVATE DATABASE command is issued.

Effect of switching user

If a user is switched within a trusted connection, no remnants of the original user are left behind. In this case, the audit policies associated with the original user are no longer considered, and the applicable audit policies are re-evaluated according to the new user. Any audit policy associated with the trusted connection is still in effect.

If a SET SESSION USER statement is used, only the session authorization ID is switched. The audit policy of the authorization ID of the original user (the system authorization ID) remains in effect and the audit policy of the new user is used as well. If multiple SET SESSION USER statements are issued within a session, only the audit policies associated with the original user (the system authorization ID) and the current user (the session authorization ID) are considered.

Data definition language restrictions

The following data definition language (DDL) statements are called AUDIT exclusive SQL statements:

- AUDIT
- CREATE AUDIT POLICY, ALTER AUDIT POLICY, and DROP AUDIT POLICY
- DROP ROLE and DROP TRUSTED CONTEXT, if the role or trusted context being dropped is associated with an audit policy

AUDIT exclusive SQL statements have some restrictions in their use:

- Each statement must be followed by a COMMIT or ROLLBACK.
- These statements cannot be issued within a global transaction, for example an XA transaction.

Only one uncommitted AUDIT exclusive DDL statement is allowed at a time across all partitions. If an uncommitted AUDIT exclusive DDL statement is executing, subsequent AUDIT exclusive DDL statements wait until the current AUDIT exclusive DDL statement commits or rolls back.

Note: Changes are written to the catalog, but do not take effect until COMMIT, even for the connection that issues the statement.

Example of auditing any access to a specific table

Consider a company where the EMPLOYEE table contains extremely sensitive information and the company wants to audit any and all SQL access to the data in that table. The EXECUTE category can be used to track all access to a table; it audits the SQL statement, and optionally the input data value provided at execution time for that statement.

There are two steps to track activity on the table. First, the security administrator creates an audit policy that specifies the EXECUTE category, and then the security administrator associates that policy with the table:

```
CREATE AUDIT POLICY SENSITIVEDATAPOLICY
    CATEGORIES EXECUTE STATUS BOTH ERROR TYPE AUDIT
COMMIT

AUDIT TABLE EMPLOYEE USING POLICY SENSITIVEDATAPOLICY
COMMIT
```

Example of auditing any actions by SYSADM or DBADM

In order to complete their security compliance certification, a company must show that any and all activities within the database by those people holding system administration (SYSADM) or database administrative (DBADM) authority can be monitored.

To capture all actions within the database, both the EXECUTE and SYSADMIN categories should be audited. The security administrator creates an audit policy that audits these two categories. The security administrator can use the AUDIT statement to associate this audit policy with the SYSADM and DBADM authorities. Any user that holds either SYSADM or DBADM authority will then have any auditable events logged. The following example shows how to create such an audit policy and associate it with the SYSADM and DBADM authorities:

```
CREATE AUDIT POLICY ADMINSPOLICY CATEGORIES EXECUTE STATUS BOTH,
   SYSADMIN STATUS BOTH ERROR TYPE AUDIT
COMMIT
AUDIT SYSADM, DBADM USING POLICY ADMINSPOLICY
COMMIT
```

Example of auditing any access by a specific role

A company has allowed its web applications access to their corporate database. The exact individuals using the web applications are unknown. Only the role that is used is known and that role is used to manage the database authorizations. The company wants to monitor the actions of anyone who is a member of that role in order to examine the requests they are submitting to the database and to ensure that they only access the database through the web applications.

The EXECUTE category contains the necessary level of auditing to track the activity of the users for this situation. The first step is to create the appropriate audit policy and associate it with the roles that are used by the web applications (in this example, the roles are TELLER and CLERK):

```
CREATE AUDIT POLICY WEBAPPPOLICY CATEGORIES EXECUTE WITH DATA
   STATUS BOTH ERROR TYPE AUDIT
COMMIT
AUDIT ROLE TELLER, ROLE CLERK USING POLICY WEBAPPPOLICY
COMMIT
```

Audit archive and extract stored procedures

The security administrator can use the SYSPROC.AUDIT_ARCHIVE stored procedure and table function, the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure, and the SYSPROC.AUDIT_LIST_LOGS table function to archive audit logs and extract data to delimited files.

The security administrator can delegate use of these routines to another user by granting the user EXECUTE privilege on these routines. Only the security administrator can grant EXECUTE privilege on these routines. EXECUTE privilege WITH GRANT OPTION cannot be granted for these routines (SQLSTATE 42501).

You must be connected to a database in order to use these stored procedures and table functions to archive or list that database's audit logs.

If you copy the archived files to another database system, and you want to use the stored procedures and table functions to access them, ensure that the database name is the same, or rename the files to include the same database name.

These stored procedures and table functions do not archive or list the instance level audit log. The system administrator must use the db2audit command to archive and extract the instance level audit log.

You can use these stored procedures and table functions to perform the following operations:

Table 79. Audit system stored procedures and table functions

Stored procedure and table function	**Operation**	**Comments**
AUDIT_ARCHIVE	Archives the current audit log.	Takes the archive path as input. If the archive path is not supplied, this stored procedure takes the archive path from the audit configuration file. The archive is run on each node, and a synchronized timestamp is appended to the name of the audit log file.
AUDIT_LIST_LOGS	Returns a list of the archived audit logs at the specified path, for the current database.	

Table 79. Audit system stored procedures and table functions (continued)

Stored procedure and table function	Operation	Comments
AUDIT_ DELIM_EXTRACT	Extracts data from the binary archived logs and loads it into delimited files.	The extracted audit records are placed in a delimited format suitable for loading into DB2 database tables. The output is placed in separate files, one for each category. In addition, the file **auditlobs** is created to hold any large objects that are included in the audit data. The file names are: • **audit.del** • **checking.del** • **objmaint.del** • **secmaint.del** • **sysadmin.del** • **validate.del** • **context.del** • **execute.del** • **auditlobs** If the files already exist, the output is appended to them. The **auditlobs** file is created if the CONTEXT or EXECUTE categories are extracted. Only archived audit logs for the current database can be extracted. Only files that are visible to the coordinator node are extracted. Only the instance owner can delete archived audit logs.

The EXECUTE category for auditing SQL statements

The EXECUTE category allows you to accurately track the SQL statements a user issues (prior to Version 9.5, you had to use the CONTEXT category to find this information).

This EXECUTE category captures the SQL statement text as well as the compilation environment and other values that are needed to replay the statement at a later date. For example, replaying the statement can show you exactly which rows a SELECT statement returned. In order to re-run a statement, the database tables must first be restored to their state when the statement was issued.

When you audit using the EXECUTE category, the statement text for both static and dynamic SQL is recorded, as are input parameter markers and host variables. You can configure the EXECUTE category to be audited with or without input values.

Note: Global variables are not audited.

The auditing of EXECUTE events takes place at the completion of the event (for SELECT statements this is on cursor close). The status that the event completed with is also stored. Because EXECUTE events are audited at completion, long-running queries do not immediately appear in the audit log.

Note: The preparation of a statement is not considered part of the execution. Most authorization checks are performed at prepare time (for example, SELECT privilege). This means that statements that fail during prepare due to authorization errors do not generate EXECUTE events.

Statement Value Index, Statement Value Type and Statement Value Data fields may be repeated for a given execute record. For the report format generated by the extraction, each record lists multiple values. For the delimited file format, multiple rows are used. The first row has an event type of STATEMENT and no values. Following rows have an event type of DATA, with one row for each data value associated with the SQL statement. You can use the event correlator and application ID fields to link STATEMENT and DATA rows together. The columns Statement Text, Statement Isolation Level, and Compilation Environment Description are not present in the DATA events.

The statement text and input data values that are audited are converted into the database code page when they are stored on disk (all audited fields are stored in the database code page). No error is returned if the code page of the input data is not compatible with the database code page; the unconverted data will be logged instead. Because each database has it's own audit log, databases having different code pages does not cause a problem.

ROLLBACK and COMMIT are audited when executed by the application, and also when issued implicitly as part of another command, such as BIND.

After an EXECUTE event has been audited due to access to an audited table, all statements that affect which other statements are executed within a unit of work, are audited. These statements are COMMIT, ROLLBACK, ROLLBACK TO SAVEPOINT and SAVEPOINT.

Savepoint ID field

You can use the Savepoint ID field to track which statements were affected by a ROLLBACK TO SAVEPOINT statement. An ordinary DML statement (such as SELECT, INSERT, and so on) has the current savepoint ID audited. However, for the ROLLBACK TO SAVEPOINT statement, the savepoint ID that is rolled back to will be audited instead. Therefore, every statement with a savepoint ID greater than or equal to that ID will be rolled back, as demonstrated by the following example. The table shows the sequence of statements run; all events with a Savepoint ID greater than or equal to 2 will be rolled back. Only the value of 3 (from the first INSERT statement) is inserted into the table T1.

Table 80. Sequence of statements to demonstrate effect of ROLLBACK TO SAVEPOINT statement

Statement	Savepoint ID
INSERT INTO T1 VALUES (3)	1
SAVEPOINT A	2
INSERT INTO T1 VALUES (5)	2
SAVEPOINT B	3

Table 80. Sequence of statements to demonstrate effect of ROLLBACK TO SAVEPOINT statement (continued)

Statement	Savepoint ID
INSERT INTO T1 VALUES (6)	3
ROLLBACK TO SAVEPOINT A	2
COMMIT	

WITH DATA option

Not all input values are audited when you specify the WITH DATA option. LOB, LONG, XML and structured type parameters appear as NULL.

Date, time, and timestamp fields are recorded in ISO format.

If WITH DATA is specified in one policy, but WITHOUT DATA is specified in another policy associated with objects involved in the execution of the SQL statement, then WITH DATA takes precedence and data is audited for that particular statement. For example, if the audit policy associated with a user specifies WITHOUT DATA, but the policy associated with a table specifies WITH DATA, when that user accesses that table, the input data used for the statement is audited.

You are not able to determine which rows were modified on a positioned-update or positioned-delete statement. Only the execution of the underlying SELECT statement is logged, not the individual FETCH. It is not possible from the EXECUTE record to determine which row the cursor is on when the statement is issued. When replaying the statement at a later time, it is only possible to issue the SELECT statement to see what range of rows may have been affected.

Example of replaying past activities

Consider in this example that as part of their comprehensive security policy, a company requires that they retain the ability to retroactively go back up to seven years to analyze the effects of any particular request against certain tables in their database. To do this, they institute a policy of archiving their weekly backups and associated log files such that they can reconstitute the database for any chosen moment in time. They require that the database audit capture sufficient information about every request made against the database to allow the replay and analysis of any request against the relevant, restored database. This requirement covers both static and dynamic SQL statements.

This example shows the audit policy that must be in place at the time the SQL statement is issued, and the steps to archive the audit logs and later to extract and analyze them.

1. Create an audit policy that audits the EXECUTE category and apply this policy to the database:

```
CREATE AUDIT POLICY STATEMENTS CATEGORIES EXECUTE WITH DATA
   STATUS BOTH ERROR TYPE AUDIT
COMMIT

AUDIT DATABASE USING POLICY STATEMENTS
COMMIT
```

2. Regularly archive the audit log to create an archive copy.

The following statement should be run by the security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT_ARCHIVE stored procedure, on a regular basis, for example, once a week or once a day, depending on the amount of data logged. These archived files can be kept for whatever period is required. The AUDIT_ARCHIVE procedure is called with two input parameters: the path to the archive directory and -2, to indicate that the archive should be run on all nodes:

```
CALL SYSPROC.AUDIT_ARCHIVE( '/auditarchive', -2 )
```

3. The security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT_LIST_LOGS table function, uses AUDIT_LIST_LOGS to examine all of the available audit logs from April 2006, to determine which logs may contain the necessary data:

```
SELECT FILE FROM TABLE(SYSPROC.AUDIT_LIST_LOGS('/auditarchive'))
   AS T WHERE FILE LIKE 'db2audit.dbname.log.0.200604%'
FILE
------------------------------------
...
db2audit.dbname.log.0.20060418235612
db2audit.dbname.log.0.20060419234937
db2audit.dbname.log.0.20060420235128
```

4. From this output, the security administrator observes that the necessary logs should be in one file: `db2audit.dbname.log.20060419234937`. The timestamp shows this file was archived at the end of the day for the day the auditors want to see.

   The security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure, uses this filename as input to AUDIT_DELIM_EXTRACT to extract the audit data into delimited files. The audit data in these files can be loaded into DB2 database tables, where it can be analyzed to find the particular statement the auditors are interested in. Even though the auditors are only interested in a single SQL statement, multiple statements from the unit of work may need to be examined in case they have any impact on the statement of interest.

5. In order to replay the statement, the security administrator must take the following actions:
   - Determine the exact statement to be issued from the audit record.
   - Determine the user who issued the statement from the audit record.
   - Recreate the exact permissions of the user at the time they issued the statement, including any LBAC protection.
   - Reproduce the compilation environment, by using the compilation environment column in the audit record in combination with the SET COMPILATION ENVIRONMENT statement.
   - Restore the database to its exact state at the time the statement was issued.

   To avoid disturbing the production system, any restore of the database and replay of the statement should be done on a second database system. The security administrator, running as the user who issued the statement, can reissue the statement as found in the statement text with any input variables that are provided in the statement value data elements.

Storage and analysis of audit logs

Archiving the audit log moves the active audit log to an archive directory while the server begins writing to a new, active audit log. Later, you can extract data from the archived log into delimited files and then load data from these files into DB2 database tables for analysis.

Configuring the location of the audit logs allows you to place the audit logs on a large, high-speed disk, with the option of having separate disks for each node in a database partitioning feature (DPF) installation. In a DPF environment, the path for the active audit log can be a directory that is unique to each node. Having a unique directory for each node helps to avoid file contention, because each node is writing to a different disk.

The default path for the audit logs on Windows operating systems is *instance*\security\auditdata and on Linux and UNIX operating systems is *instance*/security/auditdata. If you do not want to use the default location, you can choose different directories (you can create new directories on your system to use as alternative locations, if they do not already exist). To set the path for the active audit log location and the archived audit log location, use the db2audit configure command with the datapath and archivepath parameters, as shown in this example:

```
db2audit configure datapath /auditlog archivepath /auditarchive
```

The audit log storage locations you set using db2audit apply to all databases in the instance.

Note: If there are multiple instances on the server, then each instance should each have separate data and archive paths.

The path for active audit logs (datapath) in a DPF environment

In a DPF environment, the same active audit log location (set by the **datapath** parameter) must be used on each partition. There are two ways to accomplish this:

1. Use database partition expressions when you specify the **datapath** parameter. Using database partition expressions allows the partition number to be included in the path of the audit log files and results in a different path on each database partition.
2. Use a shared drive that is the same on all nodes.

You can use database partition expressions anywhere within the value you specify for the **datapath** parameter. For example, on a three node system, where the database partition number is 10, the following command:

```
db2audit configure datapath '/pathForNode $N'
```

creates the following files:

- /pathForNode10
- /pathForNode20
- /pathForNode30

Note: You cannot use database partition expressions to specify the archive log file path (**archivepath** parameter).

Archiving active audit logs

The system administrator can use the db2audit tool to archive both instance and database audit logs as well as to extract audit data from archived logs of either type.

The security administrator, or a user to whom the security administrator has granted EXECUTE privilege on the audit routines, can archive the active audit log

by running the SYSPROC.AUDIT_ARCHIVE stored procedure. To extract data from the log and load it into delimited files, they can use the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure.

These are the steps to archive and extract the audit logs using the audit routines:

1. Schedule an application to perform regular archives of the active audit log using the stored procedure SYSPROC.AUDIT_ARCHIVE.
2. Determine which archived log files are of interest. Use the SYSPROC.AUDIT_LIST_LOGS table function to list all of the archived audit logs.
3. Pass the file name as a parameter to the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure to extract data from the log and load it into delimited files.
4. Load the audit data into DB2 database tables for analysis.

The archived log files do not need to be immediately loaded into tables for analysis; they can be saved for future analysis. For example, they may only need to be looked at when a corporate audit is taking place.

If a problem occurs during archive, such as running out of disk space in the archive path, or the archive path does not exist, the archive process fails and an interim log file with the file extension `.bk` is generated in the audit log data path, for example, `db2audit.instance.log.0.20070508172043640941.bk`. After the problem is resolved (by allocating sufficient disk space in the archive path, or by creating the archive path) you must move this interim log to the archive path. Then, you can treat it in the same way as a successfully archived log.

Archiving active audit logs in a DPF environment

In a DPF environment, if the archive command is issued while the instance is running, the archive process automatically runs on every node. The same timestamp is used in the archived log file name on all nodes. For example, on a three node system, where the database partition number is 10, the following command:

```
db2audit archive to /auditarchive
```

creates the following files:

- `/auditarchive/db2audit.log.10.`*`timestamp`*
- `/auditarchive/db2audit.log.20.`*`timestamp`*
- `/auditarchive/db2audit.log.30.`*`timestamp`*

If the archive command is issued while the instance is not running, you can control on which node the archive is run by one of the following methods:

- Use the node option with the db2audit command to perform the archive for the current node only.
- Use the db2_all command to run the archive on all nodes.

  For example:

  ```
  db2_all db2audit archive node to /auditarchive
  ```

  This sets the DB2NODE environment variable to indicate on which nodes the command is invoked.

Alternatively, you can issue an individual archive command on each node separately. For example:

- On node 10:

  ```
  db2audit archive node 10 to /auditarchive
  ```

- On node 20:

  ```
  db2audit archive node 20 to /auditarchive
  ```

- On node 30:

  ```
  db2audit archive node 30 to /auditarchive
  ```

Note: When the instance is not running, the timestamps in the archived audit log file names are not the same on each node.

Note: It is recommended that the archive path is shared across all nodes, but it is not required.

Note: The AUDIT_DELIM_EXTRACT stored procedure and AUDIT_LIST_LOGS table function can only access the archived log files that are visible from the current (coordinator) node.

Example of archiving a log and extracting data to a table

To ensure their audit data is captured and stored for future use, a company needs to create a new audit log every six hours and archive the current audit log to a WORM drive. The company schedules the following call to the SYSPROC.AUDIT_ARCHIVE stored procedure to be issued every six hours by the security administrator, or by a user to whom the security administrator has granted EXECUTE privilege on the AUDIT_ARCHIVE stored procedure. The path to the archived log is the default archive path, `/auditarchive`, and the archive runs on all nodes:

```
CALL SYSPROC.AUDIT_ARCHIVE( '/auditarchive', -2 )
```

As part of their security procedures, the company has identified and defined a number of suspicious behaviors or disallowed activities that it needs to watch for in the audit data. They want to extract all the data from the one or more audit logs, place it in a relational table, and then use SQL queries to look for these activities. The company has decided on appropriate categories to audit and has associated the necessary audit policies with the database or other database objects.

For example, they can call the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure to extract the archived audit logs for all categories from all nodes that were created with a timestamp in April 2006, using the default delimiter:

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT(
     '', '', '/auditarchive',  'db2audit.%.200604%', '' )
```

In another example, they can call the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure to extract the archived audit records with success events from the EXECUTE category and failure events from the CHECKING category, from a file with the timestamp they are interested in:

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT( '', '', '/auditarchive',
    'db2audit.%.20060419034937', 'categories
     execute status success, checking status failure );
```

Audit log file names

The audit log files have names that distinguish whether they are instance-level or database-level logs and which partition they originate from in a database partitioning feature (DPF) environment. Archived audit logs have the timestamp of when the archive command was run appended to their file name.

Active audit log file names

In a DPF environment, the path for the active audit log can be a directory that is unique to each partition so that each partition writes to an individual file. In order to accurately track the origin of audit records, the partition number is included as part of the audit log file name. For example, on partition 20, the instance level audit log file name is `db2audit.instance.log.20`. For a database called testdb in this instance, the audit log file is `db2audit.db.testdb.log.20`.

In a non-DPF environment the partition number is considered to be 0 (zero). In this case, the instance level audit log file name is `db2audit.instance.log.0`. For a database called testdb in this instance, the audit log file is `db2audit.db.testdb.log.0`.

Archived audit log file names

When the active audit log is archived, the current timestamp in the following format is appended to the filename: *YYYYMMDDHHMMSS* (where YYYY is the year, MM is the month, DD is the day, HH is the hour, MM is the minutes, and SS is the seconds.

The file name format for an archive audit log depends on the level of the audit log:

instance-level archived audit log
The file name of the instance-level archived audit log is:
`db2audit.instance.log.`*`partition`*`.`*`YYYYMMDDHHMMSS`*.

database-level archived audit log
The file name of the database-level archived audit log is:
`db2audit.db`*`database`*`.log.`*`partition.YYYYMMDDHHMMSS`*.

In a non-DPF environment, the value for *partition* is 0 (zero).

The timestamp represents the time that the archive command was run, therefore it does not always precisely reflect the time of the last record in the log. The archived audit log file may contain records with timestamps a few seconds later than the timestamp in the log file name because:

- When the archive command is issued, the audit facility waits for the writing of any in-process records to complete before creating the archived log file.
- In a multi-machine environment, the system time on a remote machine may not be synchronized with the machine where the archive command is issued.

In a DPF environment, if the server is running when archive is run, the timestamp is consistent across partitions and reflects the timestamp generated at the partition at which the archive was performed.

AUDIT_LIST_LOGS table function - Lists archived audit log files

The AUDIT_LIST_LOGS table function lists the archived audit log files for a database which are present in the specified directory.

Syntax

The schema is SYSPROC.

Procedure parameters

directory
An optional input argument of type VARCHAR(1024) that specifies the directory where the archived audit file(s) will be written. The directory must exist on the server and the instance owner must be able to create files in that directory. If the argument is null or an empty string, then the search default directory is used.

Authorization

EXECUTE privilege on AUDIT_LIST_LOGS table function.

Examples

Example 1: Lists all archived audit logs in the default audit archive directory:

```
SELECT * FROM TABLE(SYSPROC.AUDIT_LIST_LOGS('')) AS T1
```

Note: This only lists the logs in the directory for database on which the query is run. Archived files have the format db2audit.db.<dbname>.log.<timestamp>

Information Returned

Table 81. The information returned for AUDIT_LIST_LOGS

Column Name	Data Type	Description
PATH	VARCHAR(1024)	Path location of the archived file.
FILE	VARCHAR(1024)	Filename of the archived file.
SIZE	BIGINT	File size of the archived file.

Chapter 27. Setting up the database environment

Considerations for Creating a Database System

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 at a location that is determined by the information you provide in the CREATE DATABASE command. If you do not specify the location of the directory path or drive when you create the database, the default location is used.

In the directory you specify as the database path in the CREATE DATABASE command, a subdirectory that uses the name of the *instance* is created. This subdirectory ensures that databases created in different instances under the same directory do not use the same path. Below the instance-name subdirectory, a subdirectory named `NODE0000` is created. This subdirectory differentiates database partitions in a logically partitioned database environment. Below the node-name directory, a subdirectory named `SQL00001` is created. This name of this subdirectory uses the database token and represents the database being created. `SQL00001` 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 directory structure appears as follows: *your_database_path*`/`*your_instance*`/NODE0000/SQL00001/`

The database directory contains the following files that are created as part of the CREATE DATABASE command.

- The files `SQLBP.1` and `SQLBP.2` contain buffer pool information. These files are duplicates of each other for backup purposes.
- The files `SQLSPCS.1` and `SQLSPCS.2` contain table space information. These files are duplicates of each other for backup purposes.
- The files `SQLSGF.1` and `SQLSGF.2` contain storage path 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 databases when automatic storage is enabled following a CREATE DATABASE *dbname* **AUTOMATIC STORAGE YES** command or ALTER DATABASE *dbname ADD STORAGE ON* statement.
- The `SQLDBCONF` file contains database configuration information. Do not edit this file.

  Note: The `SQLDBCON` file was used in previous releases and contains similar information that can be used if `SQLDBCONF` is corrupted.
  To change configuration parameters, use the UPDATE DATABASE CONFIGURATION and RESET DATABASE CONFIGURATION commands.
- 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 both history files in a text editor.

- The log control files, SQLOGCTL.LFH.1, its mirror copy SQLOGCTL.LFH.2, and SQLOGMIR.LFH, contain information about the active logs.

  Recovery processing uses information from these files to determine how far back in the logs to begin recovery. The SQLOGDIR subdirectory contains the actual log files.

  Note: You should ensure the log subdirectory is mapped to different disks than those used for your data. A disk problem could then be restricted to your data or the logs but not both. This 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, change the **newlogpath** database configuration parameter.
- The SQLINSLK file helps to ensure that a database is used by only one instance of the database manager.

At the same time a database is created, a detailed deadlocks event monitor is also created. The detailed deadlocks event monitor files are stored in the database directory of the catalog node. 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 consuming too much disk space. Removing output files that are no longer needed will allow 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 required for an operational database. Three default table spaces are created:

- 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 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)

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 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.

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 82. 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) 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 above assumes no duplicates. The space required to store an index might be over-estimated by the formula shown above.

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* + 2*L*/(*average number of keys on leaf page*)) × *pagesize*

   For DMS table spaces, add the sizes of all indexes on a table and round up to a multiple of the extent size for the table space on which the index resides.

   You should provide additional space for index growth due to INSERT/UPDATE activity, from which page splits might result.

Use the following calculation to obtain a more accurate estimate of the original index size, as well as an estimate of the number of levels in the index. (This might be of particular interest if include columns are being used in the index definition.) The average number of keys per non-leaf page is roughly:

$((.9 \times (U - (M \times 2))) \times (D + 1)) \div (K + 13 + (9 * D))$

where:

- U, the usable space on a page, is approximately equal to the page size minus 100. For a page size of 4096, U is 3996.
- D is the average number of duplicates per key value on non-leaf pages (this will be much smaller than on leaf pages, and you might want to simplify the calculation by setting the value to 0).
- $M = U \div (9 +$ *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.
- $Y = L \div N$
- Z is the number of levels in the index tree (1 initially).

Note: The calculation above applies to a single, nonpartitioned indexes, or to a single index partition for partitioned indexes.

Total number of pages is:

$T = (L + P + 2) \times 1.0002$

The additional 0.02% (1.0002) is for overhead, including space map pages.

The amount of space required to create the index is estimated as:

T × page size

Space requirements for log files

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

You will require 56 KB of space for log control files. You will also need 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.

Roll-forward recovery

If the database is enabled for roll-forward recovery, special log space requirements should be taken into consideration:

- With the **logarchmeth1** configuration parameter set to LOGRETAIN, the log files will be archived in the `log path` directory. The online disk space will eventually fill 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 logs that are waiting to be moved by the user exit program
  - New log files being formatted for future use

Circular logging

If the database is enabled for circular logging, the result of this formula is all the space that will be allocated for logging; that is, more space will not be allocated, and you will 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 will keep at least the number of active log files specified by the **logprimary** configuration parameter in the log path, therefore, you should not use the value of -1 for the **logsecond** configuration parameter in the above formula. 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 will need to 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 affects the log I/O required for a given 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.

Database partition group design

There are no database partition group design considerations if you are using a single-partition database. The DB2 Design Advisor is a tool that can be used to recommend database partition groups. The DB2 Design Advisor can be accessed from the Control Center and using db2advis from the command line processor.

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

- In a multiple partition database partition group, you can only create a unique index if it is a superset of the distribution key.
- Depending on the number of database partitions in the database, you may have one or more single-partition database partition groups, and one or more multiple partition database partition groups present.
- Each database partition must be assigned a unique number. The same database partition may be found in one or more database partition groups.
- To ensure fast recovery of the database partition containing system catalog tables, avoid placing user tables on the same database partition. This is accomplished by placing user tables in database partition groups that do not include the database partition in the IBMCATGROUP database partition group.

You should place small tables in single-partition database partition groups, except when you want to take advantage of *collocation* with a larger table. Collocation is the placement of rows from different tables that contain related data in the same database partition. Collocated tables allow DB2 Database for Linux, UNIX, and Windows to utilize more efficient join strategies. Collocated tables can reside in a single-partition database partition group. Tables are considered collocated if they reside in a multiple partition database partition group, have the same number of columns in the distribution key, and if the data types of the corresponding columns are compatible. Rows in collocated tables with the same distribution key value are placed on the same database partition. Tables can be in separate table spaces in the same database partition group, and still be considered collocated.

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

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

Distribution maps

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

A distribution map is an internally generated array containing either 32 768 entries for multiple-partition database partition groups, or a single entry for single-partition database partition groups. For a single-partition database partition group, the distribution map has only one entry containing the number of the database partition where all the rows of a database table are stored. For

multiple-partition database partition groups, the numbers of the database partition group are specified in a way such that each database partition is used one after the other to ensure an even distribution across the entire map. Just as a city map is organized into sections using a grid, the database manager uses a *distribution key* to determine the location (the database partition) where the data is stored.

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

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

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

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

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

Figure 21 shows how the row with the distribution key value (c1, c2, c3) is mapped to number 2, which, in turn, references database partition n5.

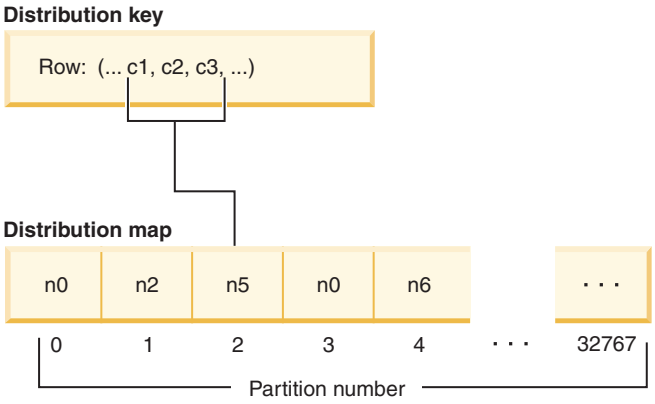


Figure 21. Data distribution using a distribution map

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

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

Distribution keys

A *distribution key* is a column (or group of columns) that is used to determine the database partition in which a particular row of data is stored. A distribution key is defined on a table using the CREATE TABLE statement. If a distribution key is not

defined for a table in a table space that is divided across more than one database partition in a database partition group, one is created by default from the first column of the primary key.

If no primary key is specified, the default distribution key is the first non-long field column defined on that table. (*Long* includes all long data types and all large object (LOB) data types). If you are creating a table in a table space associated with a single-partition database partition group, and you want to have a distribution key, you must define the distribution key explicitly. One is not created by default.

If no columns satisfy the requirement for a default distribution key, the table is created without one. Tables without a distribution key are only allowed in single-partition database partition groups. You can add or drop distribution keys later, using the ALTER TABLE statement. Altering the distribution key can only be done to a table whose table space is associated with a single-partition database partition group.

Choosing a good distribution key is important. You should take into consideration:

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

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

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

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

An inappropriate distribution key can cause uneven data distribution. Columns with unevenly distributed data, and columns with a small number of distinct values should not be chosen as distribution keys. The number of distinct values must be great enough to ensure an even distribution of rows across all database partitions in the database partition group. The cost of applying the distribution algorithm is proportional to the size of the distribution key. The distribution key cannot be more than 16 columns, but fewer columns result in better performance. Unnecessary columns should not be included in the distribution key.

The following points should be considered when defining distribution keys:

- Creation of a multiple-partition table that contains only BLOB, CLOB, DBCLOB, LONG VARCHAR, LONG VARGRAPHIC, XML, or structured data types is not supported.
- The distribution key definition cannot be altered.
- The distribution key should include the most frequently joined columns.
- The distribution key should be made up of columns that often participate in a GROUP BY clause.

- Any unique key or primary key must contain all of the distribution key columns.
- In an online transaction processing (OLTP) environment, all columns in the distribution key should participate in the transaction by using equal (=) predicates with constants or host variables. For example, assume you have an employee number, *emp_no*, that is often used in transactions such as:

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

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

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

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

Table collocation

You may discover that two or more tables frequently contribute data in response to certain queries. In this case, you will want related data from such tables to be located as close together as possible. In an environment where the database is physically divided among two or more database partitions, there must be a way to keep the related pieces of the divided tables as close together as possible. The ability to do this is called *table collocation*.

Tables are collocated when they are stored in the same database partition group, and when their distribution keys are compatible. Placing both tables in the same database partition group ensures a common distribution map. The tables may be in different table spaces, but the table spaces must be associated with the same database partition group. The data types of the corresponding columns in each distribution key must be *partition-compatible*.

DB2 Database for Linux, UNIX, and Windowssoftware can recognize, when accessing more than one table for a join or a subquery, that the data to be joined is located at the same database partition. When this happens, DB2 can perform the join or subquery at the database partition where the data is stored, instead of having to move data between database partitions. This ability has significant performance advantages.

Partition compatibility

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

Partition-compatibility has the following characteristics:

- A base data type is compatible with another of the same base data type.

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

Replicated materialized query tables

A *materialized query table* is a table that is 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 could be resolved using a materialized query table, the query may be rewritten to use the materialized query table.

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

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

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

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

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.

Storage flexibility
With DMS and SMS table spaces, you can specify which storage devices are used to store data. You could choose, for example, choose to store current, operational data in table spaces that reside on faster devices, and historical data in table spaces that reside on slower (and less expensive) devices.

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 22 on page 314 illustrates the relationship between tables and table spaces within a database, and the containers associated with that database.

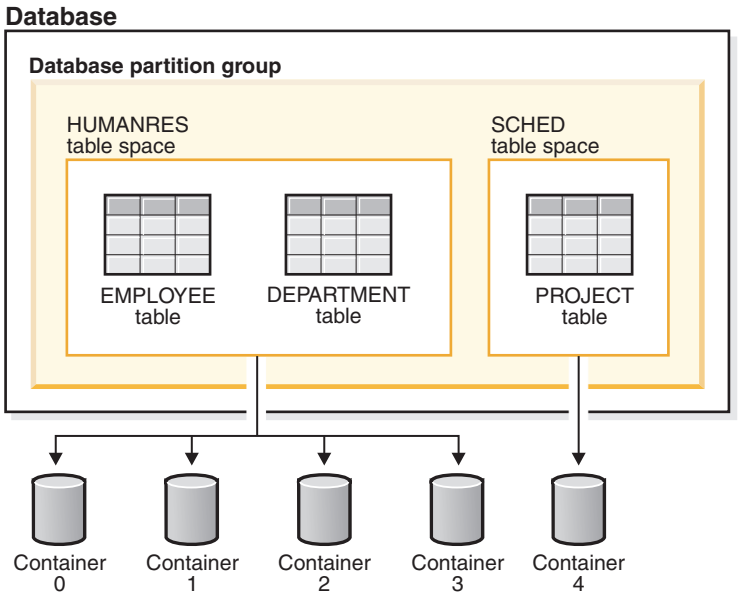


Figure 22. Table spaces and tables in a database

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 23 on page 315 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.

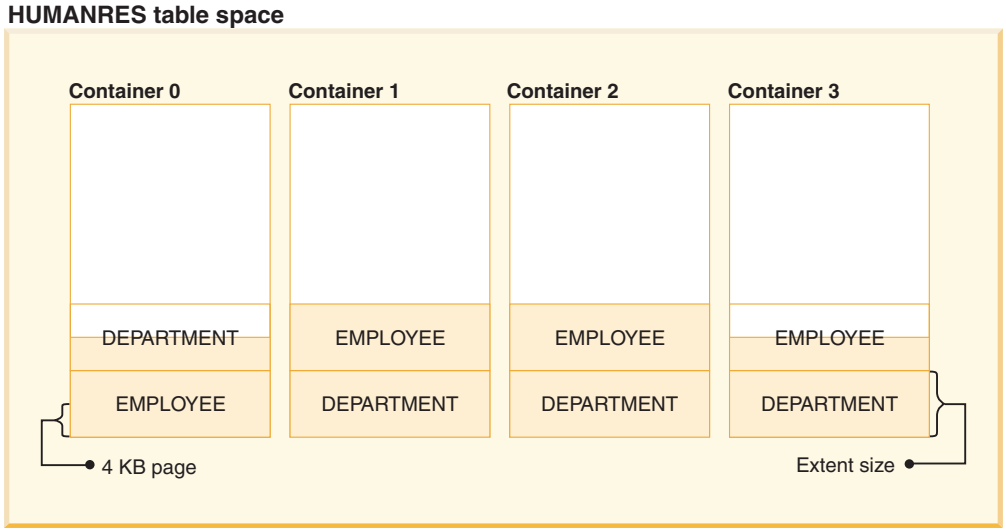


Figure 23. Containers and extents in a table space

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. Unlike database managed (DMS) table spaces, storage space is not pre-allocated when the table space is created; it 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.

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) will be allocated when space is required. Otherwise, space will be allocated one page at a time.

Multi-page file allocation only affects 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 has been 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 has

been 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 very 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 will create 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 will be 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 (SQLnnnnn.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.

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 overhead. 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 1061.
- 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 24 shows how extents are allocated in a logical table space address map.

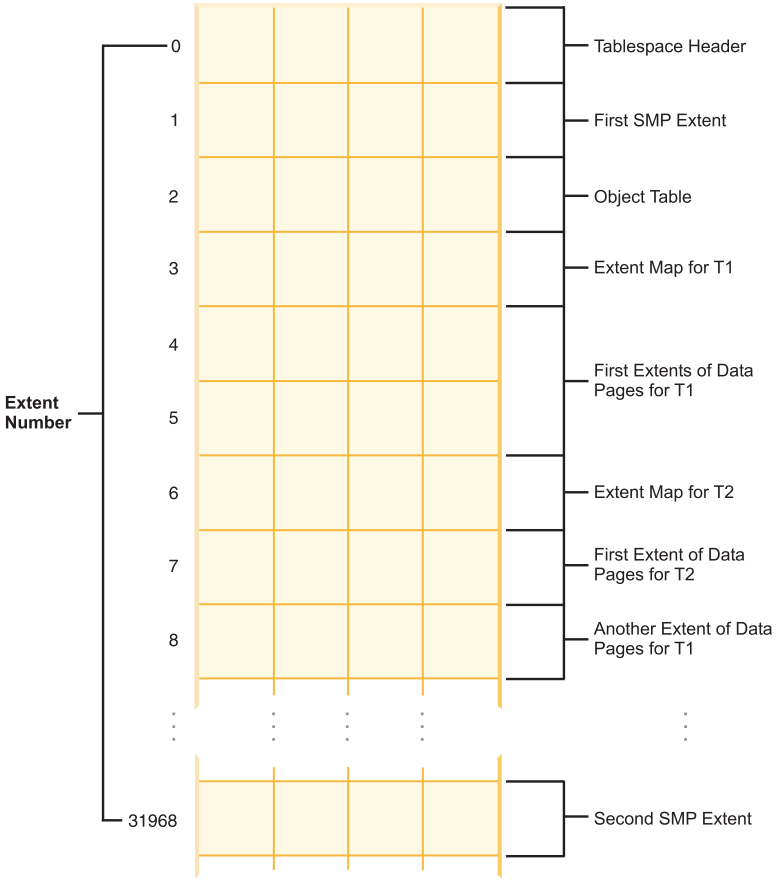


Figure 24. Logical table space address map

Comparison of SMS and DMS table spaces

There are a number of trade-offs to consider when determining which type of table space you should use to store your data.

Advantages of an SMS Table Space:

- Space is not allocated by the system until it is required, thus data may not be held in contiguous pages.
- Creating a table space requires less initial work, because you do not have to predefine the containers

- Indexes created on range partitioned data can be stored in a different table space than the table data

Advantages of a DMS Table Space:

- The size of a table space can be increased by adding or extending containers, using the ALTER TABLESPACE statement. Existing data can be automatically rebalanced across the new set of containers to retain optimal I/O efficiency.
- Data can be split across multiple table spaces, based on the type of data being stored:
  - Long field (LF) and large object (LOB) data
  - Indexes
  - Regular table data

  You might want to separate your table data for performance reasons, or to increase the amount of data stored for a table. For example, if you are using large table spaces with a 4 KB page size, you can have a table with 8 TB of regular table data, a separate table space with 8 TB of index data, and another separate table space with 8 TB of long data. If these three types of data were stored in one table space instead, the total space would be limited to 8 TB. Using larger page sizes allows you to store even more data. See the related links for the complete list of database manager page size limits.
- Indexes created on range partitioned data can be stored in a different table space than the table data.
- The location of the data on the disk can be controlled, if this is allowed by the operating system.
- In general, a well-tuned set of DMS table spaces will outperform SMS table spaces.

Note: For performance-sensitive applications, particularly those involving a large number of insert operations, it is suggested that you use DMS table spaces.

Also, placement of data can differ on the two types of table spaces. For example, consider the need for efficient table scans: it is important that the pages in an extent are physically contiguous. With SMS, the file system of the operating system decides where each logical file page is physically placed. The pages might be allocated contiguously depending on the level of other activity on the file system and the algorithm used to determine placement. With DMS, however, the database manager can ensure the pages are physically contiguous because it interfaces with the disk directly.

In general, small personal databases are easiest to manage with SMS table spaces. On the other hand, for large, growing databases you will probably only want to use SMS table spaces for the temporary table spaces and catalog table space, and separate DMS table spaces, with multiple containers, for each table. In addition, you will probably want to store long field (LF) data and indexes on their own table spaces.

If you choose to use DMS table spaces with device containers, you must be willing to tune and administer your environment.

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.

Before Creating the Database

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 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)
```

About this task

To start the instance using the command line, enter:

```
db2start
```

Note: When you run commands to start or stop an instance's database manager, the DB2 database manager applies the command to the current 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 from db2start, 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

To start an instance using the command line, enter:

```
db2start
```

Note: When you run commands to start or stop an instance's database manager, the DB2 database manager applies the command to the current instance.

The db2start command will launch 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" switch when invoking db2start. The DB2 database instance can also be started as a service 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 can not use the ″/D″ switch to start a DB2 instance as a process in a partitioned database environment.

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.

Stopping instances (Linux, UNIX)

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

Before you begin

To stop an instance on your Linux or UNIX system, you must do the following:

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, list applications. You need SYSADM, SYSCTRL, or SYSMAINT authority for this.
3. Force all applications and users off the database. You require SYSADM or SYSCTRL authority to force users.

About this task

The db2stop command can only be run 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.

Note: 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. The db2stop command stops the instance defined by the DB2INSTANCE environment variable.

To stop the instance using the command line, enter:

```
db2stop
```

You can use the db2stop command to stop, or drop, individual database partitions within a partitioned database environment. When working in a partitioned database environment and you are attempting to drop a logical partition using

```
db2stop drop nodenum <0>
```

You must ensure that no users are attempting to access the database. If they are, you will receive an error message SQL6030N.

Note: When you run commands to start or stop an instance's database manager, the DB2 database manager applies the command to the current instance. For more information, see "Setting the current instance environment variables" on page 1063.

Stopping instances (Windows)

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

Before you begin

To stop an instance on your system, you must do the following:

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, list applications. You need SYSADM, SYSCTRL, or SYSMAINT authority for this.
3. Force all applications and users off the database. You require SYSADM or SYSCTRL authority to force users.

About this task

The db2stop command can only be run 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.

Note: 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. The db2stop command stops the instance defined by the DB2INSTANCE environment variable.

To stop an instance on your system, use one of the following methods:

- Stop using the db2stop command.
- Stop using the NET STOP command.
- Stop the instance from within an application.

Recall that when you are using the database manager in a partitioned database environment, each database partition server is started as a service. Each service must be stopped.

Note: When you run commands to start or stop an instance's database manager, the database manager applies the command to the current instance. For more information, see "Setting the current instance environment variables" on page 1063.

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).

Working with 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.

Managing licenses

The management of licenses for your DB2 database products is done primarily through the License Center within the Control Center of the online interface to the product. From the License Center you can check the license information, statistics, and current users for each of the installed products.

In addition to the License Center, the command line licensing tool db2licm can be used to perform license functions. With this command, you are able to add, remove, list, and modify licenses and policies installed on your local system.

To assist you in managing your licenses, a compliance report lists the compliance or noncompliance of DB2 features with your current product entitlement. To be in compliance with your license agreement, you should apply the license key. However, even without applying a license key, your DB2 database product will continue to operate without interruption or restriction, unless you installed a DB2 database product trial image or a DB2 fix pack installation image. A trial DB2 database product will stop working after a 90 day trial period. If you installed a DB2 fix pack installation image over an existing DB2 database product acquired from Passport Advantage®, the fix pack installation image will continue to operate uninterrupted.

DB2 database product trial images include access to all features available in the edition you are using. You can download a DB2 try and buy image from Trials and demos.

If you installed a DB2 database product with a trial license and now want to upgrade to a full license, you do not need to reinstall the DB2 database product. You simply upgrade your license.

Note: The trial license for DB2 Enterprise Server Edition on 32-bit Linux cannot be upgraded to a full production license.

Additional considerations for partitioned database environments

Setting up partitioned database environments

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

About this task

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

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

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

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

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

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

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

When the first database in the system is created, a system database directory is formed. It is appended with information about any other databases that you create. When working on UNIX, the system database directory is `sqldbdir` and is located in the `sqllib` directory under your home directory, or under the directory where DB2 database was installed. When working on UNIX, this directory must reside on a shared file system, (for example, NFS on UNIX platforms) because there is only one system database directory for all the database partitions that make up the partitioned database environment. When working on Windows, the system database directory is located in the instance directory.

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

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

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

Creating node configuration files

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

About this task

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

Windows considerations

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

Note: You should not create files or directories under the `sqllib` subdirectory other than those created by the database manager to prevent the loss of data if an instance is deleted. There are two exceptions. If your system supports stored procedures, put the stored procedure applications in the function subdirectory

under the sqllib subdirectory. The other exception is when user-defined functions (UDFs) have been created. UDF executables are allowed in the same directory.

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

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

Tokens are delimited by blanks. The variables are:

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

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

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

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

NODE *nnnn*

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

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

(There is an exception when netname is specified. In this situation, netname is used for most communications, with hostname only being used 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 hostname, 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 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 hostname.

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 83. 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.

Enabling communication between database partitions using FCM communications

In a partitioned database environment, most communication between database partitions is handled by the fast communications manager (FCM).

To enable the FCM at a database partition and allow communication with other database partitions, you must create a service entry in the database partition's `services` file of the `etc` directory as shown below. The FCM uses the specified port to communicate. If you have defined multiple database partitions on the same host, you must define a range of ports as shown below.

Before attempting to manually configure memory for the fast communications manager (FCM), it is recommended that you start with the automatic setting, which is also the default setting, for the number of FCM Buffers (*fcm_num_buffers*) and for the number of FCM Channels (*fcm_num_channels*). Use the system monitor data for FCM activity to determine if this setting is appropriate.

Windows Considerations

If you are using DB2 Enterprise Server Edition in the Windows environment, the TCP/IP port range is automatically added to the services file by:

- The install program when it creates the instance or adds a new database partition
- The db2icrt utility when it creates a new instance
- The db2ncrt utility when it adds the first database partition on the computer

The syntax of a service entry is as follows:

```
DB2_instance port/tcp #comment
```

DB2_*instance*
The value for *instance* is the name of the database manager instance. All characters in the name must be lowercase. Assuming an instance name of `db2puser`, you would specify `DB2_db2puser`

port/**tcp**
The TCP/IP port that you want to reserve for the database partition.

#*comment*
Any comment that you want to associate with the entry. The comment must be preceded by a pound sign (#).

If the `services` file of the `etc` directory is shared, you must ensure that the number of ports allocated in the file is either greater than or equal to the largest number of multiple database partitions in the instance. When allocating ports, also ensure that you account for any processor that can be used as a backup.

If the `services` file of the `etc` directory is not shared, the same considerations apply, with one additional consideration: you must ensure that the entries defined for the DB2 database instance are the same in all `services` files of the `etc` directory (though other entries that do not apply to your partitioned database environment do not have to be the same).

If you have multiple database partitions on the same host in an instance, you must define more than one port for the FCM to use. To do this, include two lines in the `services` file of the `etc` directory to indicate the range of ports you are allocating. The first line specifies the first port, while the second line indicates the end of the block of ports. In the following example, five ports are allocated for the instance `sales`. This means no processor in the instance has more than five database partitions. For example,

```
DB2_sales         9000/tcp
DB2_sales_END     9004/tcp
```

Note: You must specify `END` in uppercase only. Also you must ensure that you include both underscore (_) characters.

FCM considerations for a Common Criteria compliant environment

When the Database Partitioning Feature (DPF) is used with multiple physical nodes (that is, two or more physically separate computers), communication between the physical nodes occurs over a network using TCP/IP. The DB2 database manager does not protect that communication (that is, it is not encrypted or otherwise secured).

Therefore, the DB2 database manager should be installed and operated in an environment where such protections are provided by the hardware or operating system associated with the DB2 instance. For example, DPF-related network traffic could be isolated to a separate network (wholly contained in the same secure environment where the DB2 servers reside), or encrypted either using IPsec or through the use of a hardware based network encryption solution.

Creating a Database and Database Objects

Creating databases

You create a database using the CREATE DATABASE command. To create a database from a client application, call the sqlecrea API.

Before you begin

Before you begin

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. Once it has been created and populated with data, changes can be made. However depending on how you set up the database initially, it will likely require more effort and make your data unavailable for use while the changes are being 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 on the **RESTRICTIVE** option, see the CREATE DATABASE command.

About this task

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 personl
    WITH "Personnel DB for BSchiefer Co"
  ```

What to do next

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.

See "Automatic features" on page 1059 for other features that are enabled by default when you create a database.

Event Monitor

At the same time a database is created, a detailed deadlocks event monitor is also created. As with any monitor, there is some overhead associated with this event monitor. If you do not want the detailed deadlocks event monitor, then the event monitor can be dropped 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 have the ability to 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 desired code set and territory when creating the database. See the CREATE DATABASE command or the sqlecrea API for information on setting the code set and territory.

Initial database partition groups

When a database is initially created, database partitions are created for all database partitions specified in the db2nodes.cfg file. Other database partitions can be added or removed with the ADD DBPARTITIONNUM and DROP DBPARTITIONNUM VERIFY commands.

Three database partition groups are defined:

- IBMCATGROUP for the SYSCATSPACE table space, holding system catalog tables
- IBMTEMPGROUP for the TEMPSPACE1 table space, holding temporary tables created during database processing
- IBMDEFAULTGROUP for the USERSPACE1 table space, by default holding user tables and indexes.

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 SYSTEM USING ('path')
    EXTENTSIZE value PREFETCHSIZE value
  USER TABLESPACE
    MANAGED BY DATABASE USING (FILE 'path' 5000,
                               FILE 'path' 5000)
    EXTENTSIZE value PREFETCHSIZE value
  TEMPORARY TABLESPACE
    MANAGED BY SYSTEM USING ('path')
  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 SYSTEM USING ('d:\pcatalog','e:\pcatalog')
    EXTENTSIZE 16 PREFETCHSIZE 32
  USER TABLESPACE
    MANAGED BY DATABASE USING (FILE'd:\db2data\personl' 5000,
                               FILE'd:\db2data\personl' 5000)
    EXTENTSIZE 32 PREFETCHSIZE 64
  TEMPORARY TABLESPACE
    MANAGED BY SYSTEM USING ('f:\db2temp\personl')
  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.

System catalog tables

A set of system catalog tables is created and maintained for each database. These tables contain information about the definitions of the database objects (for example, tables, views, indexes, and packages), and security information about the type of access that users have to these objects. These tables are stored in the SYSCATSPACE table space.

These tables are updated during the operation of a database; for example, when a table is created. You cannot explicitly create or drop these tables, but you can query and view their content. When the database is created, in addition to the system catalog table objects, the following database objects are defined in the system catalog:

- A set of routines (functions and procedures) in the schemas SYSIBM, SYSFUN, and SYSPROC.
- A set of read-only views for the system catalog tables is created in the SYSCAT schema.
- A set of updatable catalog views is created in the SYSSTAT schema. These updatable views allow you to update certain statistical information to investigate the performance of a hypothetical database, or to update statistics without using the RUNSTATS utility.

After your database has been created, you might want to limit the access to the system catalog views.

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.

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. Refer to ″DB2 CLI bind files and package names″ for details.

To bind or rebind the utilities to a database, from the command line, invoke the following commands, where `sample` is the name of the database:

```
connect to sample
bind @db2ubind.lst
```

Creating database partition groups

You create a database partition group with the CREATE DATABASE PARTITION GROUP statement. This statement specifies the set of database partitions on which the table space containers and table data are to reside.

Before you begin

The computers and systems must be available and capable of handling a partitioned database environment. You have purchased and installed DB2 Enterprise Server Edition. The database must exist.

About this task

This statement also:

- Creates a distribution map for the database partition group.
- Generates a distribution map ID.
- Inserts records into the following catalog tables:
  - SYSCAT.DBPARTITIONGROUPS
  - SYSCAT.PARTITIONMAPS
  - SYSCAT.DBPARTITIONGROUPDEF

To create a database partition group using the Control Center:

1. Expand the object tree until you see the **Database partition groups** folder.
2. Right-click the **Database partition groups** folder, and select **Create** from the pop-up menu.
3. On the Create Database partition groups window, complete the information, use the arrows to move database partitions from the Available database partitions box to the Selected database partitions box, and click **OK**.

To create a database partition group using the command line, enter:

```
CREATE DATABASE PARTITION GROUP db-partition-group-name
   ON DBPARTITIONNUM (db-partition-number1,db-partition-number1)
```

or

```
CREATE DATABASE PARTITION GROUP db-partition-group-name
   ON DBPARTITIONNUMS (db-partition-number1
   TO db-partition-number2)
```

For example, assume that you want to load some tables on a subset of the database partitions in your database. You would use the following command to create a database partition group of two database partitions (1 and 2) in a database consisting of at least three (0 to 2) database partitions:

```
CREATE DATABASE PARTITION GROUP mixng12 ON DBPARTITIONNUM (1,2)
```

or

```
CREATE DATABASE PARTITION GROUP mixng12 ON DBPARTITIONNUMS (1 TO 2)
```

The CREATE DATABASE command or sqlecrea() API also create the default system database partition groups, IBMDEFAULTGROUP, IBMCATGROUP, and IBMTEMPGROUP.

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

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.

Procedure

- To create an SMS table space using the command line, enter:

  ```
  CREATE TABLESPACE name
        MANAGED BY SYSTEM
        USING ('path')
  ```

- 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

- 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 statements above is equivalent; table spaces created in such a database will, by default, be automatic storage table spaces unless you specify otherwise.

Example

Example 1: 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 2: 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 700. This means that only the instance owner has read, write, and execute access. Because only the instance owner has this access, the following scenario might occur when multiple instances are being created:

- Using the same directory structure as described above, 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 3: 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 4: 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 5: 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.

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.

Note: The PREFETCHSIZE value should be a multiple of the EXTENTSIZE value. For example if the EXTENTSIZE is 10, the PREFETCHSIZE should be 20 or 30. You should also consider letting the database manager automatically determine the prefetch size, by setting PREFETCHSIZE to AUTOMATIC.

Direct I/O (DIO) improves memory performance because it bypasses caching at the file system level. This process reduces CPU overhead and makes more memory available to the database instance.

Concurrent I/O (CIO) includes the advantages of DIO and also relieves the serialization of write accesses.

DIO and CIO are supported on AIX; DIO is supported on HP-UX, Solaris, Linux, and Windows operating systems.

The keywords NO FILE SYSTEM CACHING and FILE SYSTEM CACHING are part of the CREATE and ALTER TABLESPACE SQL statements to allow you to specify whether DIO or CIO is to be used with each table space. When NO FILE SYSTEM CACHING is in effect, the database manager attempts to use Concurrent I/O (CIO) wherever possible. In cases where CIO is not supported (for example, if JFS is used), DIO is used 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.

You might want to disable this feature if you plan to run numerous drop table operations, and you either use circular logging or you do not think you will want to recover any of the dropped tables. To disable this feature, you can explicitly set the DROPPED TABLE RECOVERY option to OFF when you issue the CREATE TABLESPACE statement. Alternatively, you can turn off the dropped table recovery feature for an existing table space using the ALTER TABLESPACE statement.

Table spaces in database partition groups

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

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

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.

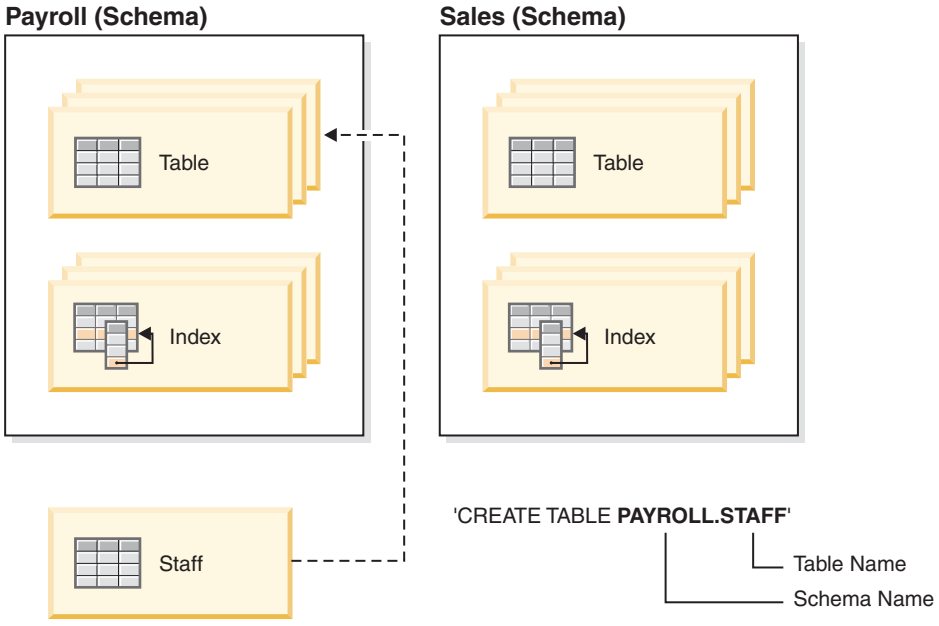
Schemas can be compared to directories, with the current schema being the current directory. Using this analogy, SET SCHEMA is equivalent to the change directory command.

Important: It is important to understand that there is no relation between authorization IDs and schemas except for the default CURRENT SCHEMA setting (described below).

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 1061.

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.

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.

About this task

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 1061.

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, whereas partitioned tables let you spread data across more than one database partition.

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.

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 utilizing the predetermined record identifier (RID) aspect of RCT tables.

Temporary tables

These types of tables are used as temporary work tables for a variety of 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 will be loosely clustered (not monotonically increasing), consider using a regular table and indexes. If you have data records that will have duplicate (but not unique) values in the key, you should 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, you should 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 above, 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 exploit 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 1061.

Creating tables

The database manager controls changes and access to the data stored in the tables. You can create tables 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 quite simple.

Example

```
CREATE TABLE <table name> (<column name> <data type> <column options>,
                           (<column name> <data type> <column options>, ...)
```

The `<table name>` may or may not include a qualifier. The name must be unique when compared to all table, view, and alias names in the system catalog. The name must also not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT.

The `<column name>` names the columns in the table. This name cannot be qualified and must be unique within the other columns of the table.

Any <column options> that exist for a column further define the attributes of the column. The options include NOT NULL in order to prevent the column from containing null values, specific options for LOB data types, and the SCOPE of the reference type columns, any constraints on the columns, and any defaults for the columns. For more information, see the CREATE TABLE statement.

Data types for columns

When defining columns, you need to name the columns, define the type of data that will be included in those columns (called *data types*), and define the length of the data for each column in the table you are creating.

About this task

Character data stored as binary data

Small integer
This data type is used to store binary integer values that have a precision of 15 bits. The range for a small integer value is -32 768 to +32 767. The small integer data type uses the smallest amount of storage space possible to store numerical values (2 bytes of space is required for each value stored). The term SMALLINT is used to declare a small integer column in a table definition.

Integer
This data type is used to store binary integer values that have a precision of 31 bytes. Although the integer data type requires twice as much storage space as the small integer data type (4 bytes of space is required for each value stored), its range of values is much greater. The range for an integer value is -2 147 483 648 to +2 147 483 647. The terms INTEGER and INT can be used to declare an integer column in a table definition.

Big integer
This data type is used to store binary integer values that have a precision of 63 bits on platforms that provide support for 64 bit integers. Processing large numbers that are stored as big integers is more efficient than processing similar numbers that have been stored as decimal values. In addition, calculations performed with big integer values are more precise than calculations performed with real or double values.

This data type requires four times as much storage space as the small integer data type (8 bytes of space is required for each value stored.) The range for a big integer is -9 223 372 036 854 775 808 to +9 223 372 036 854 775 807. The term BIGINT is used to declare a big integer column in a table definition.

Decimal
This data type is used to store numbers that contain both whole and fractional parts; the parts are combined and stored in packed decimal format. A precision (the total number of digits) and a scale (the number of digits to use for the fractional part of the number) must be specified whenever a decimal data type is declared. The range for the precision of a decimal is 1 to 31. The amount of storage space needed to store a decimal value can be calculated by using the following equation: *precision* divided by 2 (truncated) + 1 = *bytes of space required*.

For example, a DECIMAL(8,2) value would require 5 bytes of storage space (8 divided by 2 = 4; 4 + 1 = 5), whereas a DECIMAL(7,2) value would require 4 bytes of storage space (7 divided by 2 = 3.5 (truncated to 3); 3 + 1 = 4).

The terms DECIMAL, DEC, NUMERIC, and NUM can all be used to declare a decimal column in a table definition.

Note: If the precision and scale values are not provided for a decimal column definition, by default, a precision value of 5 and a scale value of 0 are used (therefore, 3 bytes of storage space is needed).

Single-precision floating point

This data type is used to store a 32-bit approximation of a real number. Although the single-precision floating-point data type and the integer data type require the same amount of storage space (4 bytes of space is required for each value stored), the range for a single-precision floating-point number is much greater: $10E^{-38}$ to $10E^{+38}$.

The terms REAL and FLOAT can be used to declare a single-precision floating-point column in a table definition. However, if the term FLOAT is used, the length specified for the column must be between 1 and 24–the FLOAT can be used to represent both single- and double-precision floating-point data types; the length specified determines which actual data type is to be used.

Double-precision floating point

The double-precision floating-point data type is used to store a 64-bit approximation of a real number. Although the double-precision floating-point data type requires the same amount of storage space as the big integer data type (8 bytes of space is required for each value stored), the range for a double-precision floating-point number is the largest possible: -1.79760^{+308} to $-2.225E^{-307}$, 0, and $2.225E^{-307}$ to $-1.79769E^{+308}$.

Decimal-precision floating point

A decimal floating-point data type that is useful in business applications (for example, financial applications) that deal with exact decimal values. Binary floating-point data types (REAL and DOUBLE), which provide binary approximations for decimal data, are not appropriate in such applications. DECFLOAT combines the accuracy of DECIMAL with some of the performance advantage of FLOAT, which is beneficial in applications where monetary values are being manipulated.

Fixed-length character string

This data type is used to store character and character string data that has a specific length that does not exceed 254 characters. The terms CHARACTER and CHAR can be used to declare a fixed-length character string column in a table definition; the length of the character string data to be stored must be specified whenever a fixed-length character string data type is declared. The amount of storage space needed to store a fixed-length character string value can be determined by using the following equation: *fixed length* x 1 = *bytes of space required*. For example, a CHAR(8) value would require 8 bytes of storage.

Note: When fixed-length character string data types are used, storage space can be wasted if the actual length of the data is significantly smaller than the length specified when the column was defined. For example, if the values YES and NO were to be stored in a column that was defined as CHAR(20). Therefore, the fixed length specified for a fixed-length character string column should be as close as possible to the actual length of the data that will be stored in the column.

Variable length character data

This data type is used to store character string data that varies in length. Varying-length character string data can be up to 32 672 characters long; however, the actual length allowed is governed by one restriction: the data must fit on a single table space page. This means that for a table that resides in a table space that used 4K pages, varying-length character string data cannot be more than 4 092 characters long; for a table that resides in a table space that used 8K pages, varying-length character string data cannot be more than 8 188 characters long and so on, up to 32K. Because table spaces are created with 4K pages by default, you must explicitly create a table space with a larger page size if you want to use a varying length character string data type to store strings that contain more than 4 092 characters.

Note:

- You must also have sufficient space in the table row to accommodate the character string data. In other words, the storage requirements for other columns in the table must be added to the storage requirements of the character string data and the total amount of storage space needed must not exceed the size of the table space's page.
- When a varying-length string data value is updated and the new value is larger than the original value, the record containing the value will be moved to another page in the table. Such records are known as pointer records. Too many pointer records can cause a significant decrease in performance because multiple pages must be retrieved in order to process a single data record.

The terms CHARACTER VARYING, CHAR VARYING, and VARCHAR can be used to declare a varying-length character string column in a table definition. When a varying length character string column is defined, the maximum number of characters that are expected to be stored in that column must be specified as part of the declaration. Subsequent character string data values that are stored in the column can be shorter than or equal to the maximum length specified; if they are longer, they will not be stored and error is returned.

The amount of storage space needed to store a varying-length character string value can be determined by using the following equation: (*string length* x 1) + 4 = *bytes of space required*. Thus, if a character string containing 30 characters were stored using a VARCHAR(30) data type, that particular value would require 34 bytes of storage space. (All character strings using this data type would have to be less than or equal to 30 characters in length.)

Variable length long character data

The varying-length long character string data type is also used to store string data that varies in length. This data type is used to store character string data that is less than or equal to 32 700 characters long in a table

that resides in a table space that uses 4K pages. In other words, when the varying-length long character string is used, the page size/character string data length restrictions that apply to varying-length character string data are not applicable.

The term LONG VARCHAR is used to declare a varying-length long character string column in a table definition. The amount of storage space needed to store a varying-length character string value can be determined by using this equation: (*string length* x 1) + 24 = *bytes of space required*. The LONG VARCHAR and LONG VARGRAPHIC data types are deprecated and might be removed in a future release. When choosing a data type for a column, use data types such as VARCHAR, VARGRAPHIC, CLOB, or DBCLOB since these will continue to be supported in future releases and are recommended for portable applications.

Note: The FOR BIT DATA clause can be used with any character string data type when declaring a column in a table definition. If this clause is used, code page conversions will not be performed during data exchange operations and the data itself will be treated and compared as binary (bit) data.

Character large objects (CLOBS)

A CLOB (character large object) value can be up to 2 gigabytes (2 147 483 647 bytes) long. A CLOB is used to store large SBCS or mixed (SBCS and MBCS) character-based data (such as documents written with a single character set) and, therefore, has an SBCS or mixed code page associated with it.

Variable length character stored as binary data (Large objects–LOBS and Binary large objects–BLOBs)

The term *large object* and the generic acronym LOB refer to the BLOB, CLOB, or DBCLOB data type. LOB values are subject to restrictions that apply to LONG VARCHAR values, as described in the section "Variable length character data". These restrictions apply even if the length attribute of the LOB string is 254 bytes or less. This data type is used to store binary string data that varies in length. It is frequently used to store nontraditional data such as documents, graphic images, pictures, audio, and video.

Note: Binary large objects data cannot be manipulated by SQL the same way that other data can. For example, binary large object values cannot be sorted.

Unicode data

All data types supported are also supported in a Unicode database. In particular, graphic string data is supported for a Unicode database, and is stored in UCS-2 encoding. Every client, including SBCS clients, can work with graphic string data types in UCS-2 encoding when connected to a Unicode database.

DATE The DATE data type is used to store a three-part value (year, month, and day) that designates a valid calendar data. The range for the year part is 0001 to 9999; the range for the month part is 1 to 12; and the range for the day part is 1 to n (28, 29, 30, or 31) where n is dependent upon the month part and whether the year part corresponds to a leap year. Externally, the date data type appears to be a fixed-length character string data type that has a length of 10. However, internally, the date data type requires much less storage space–4 bytes of space is required for each value stored,

because date values are stored as packed strings. The term DATE is used to declare a date column in a table definition.

Like dates, the representation of time varies in different parts of the world. The format of a time value is determined by the territory code associated with the database being used. Table 84 shows the time formats that are available, along with an example of their string representation:

Table 84. Date formats (YYYY = Year, MM = Month, DD = Day)

Format name	Abbreviation	Date string format
International Standards Organization	ISO	YYYY-MM-DD
IBM USA Standard	USA	MM/DD/YYYY
IBM European Standard	EUR	MM/DD/YYYY
Japanese Industrial Standard	JIS	YYYY-MM-DD
Site specific	LOC	Based on database's territory code

TIME

The TIME data type is used to store a three-part value (hours, minutes, and seconds) that designates a valid time of day under a 24-hour clock. The range for the hours part is 0 to 24; the range for the minutes part is 0 to 59; and the range for the seconds part is also 0 to 59. (If the hours part is set to 24, the minutes and seconds parts must be set to 0.) Externally, the time data type appears to be a fixed-length character string data type that has a length of 8. However, like date values, time values are stored as packed strings–in this case, 3 bytes of space is required for each time value stored. The term TIME is used to declare a time column in a table definition.

TIMESTAMP

A TIMESTAMP is a six or seven-part value (year, month, day, hour, minute, second, and optional fractional seconds) designating a date and time as defined above, except that the time could also include an additional part designating a fraction of a second. The number of digits in the fractional seconds is specified using an attribute in the range from 0 to 12 with a default of 6.

Numeric data

All numbers have a sign and a precision. The precision is the number of bits or digits excluding the sign. See the data type section in the description of the CREATE TABLE statement.

XML data

The XML data type is used to define columns of a table that store XML values, where all stored XML values must be well-formed XML documents. The introduction of this native XML data type provides the ability to store well-formed XML documents in their native hierarchical format in the database alongside other relational 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
```

Tables in partitioned database environments

There are performance advantages to creating a table across several database partitions in a partitioned database environment. The work associated with the retrieval of data can be divided among the database partitions.

Before you begin

Before creating a table that will be physically divided or distributed, you need to consider the following:

- Table spaces can span more than one database partition. The number of database partitions they span depends on the number of database partitions in a database partition group.
- Tables can be collocated by being placed in the same table space or by being placed in another table space that, together with the first table space, is associated with the same database partition group.

About this task

Creating a table that will be a part of several database partitions is specified when you are creating the table. There is an additional option when creating a table in a partitioned database environment: the *distribution key*. A distribution key is a key that is part of the definition of a table. It determines the database partition on which each row of data is stored.

If you do not specify the distribution key explicitly, the following defaults are used. *Ensure that the default distribution key is appropriate.*

- If a primary key is specified in the CREATE TABLE statement, the first column of the primary key is used as the distribution key.
- For a multiple partition database partition group, if there is no primary key, the first column that is not a long field is used.
- If no columns satisfy the requirements for a default distribution key, the table is created without one (this is allowed only in single-partition database partition groups).

You must be careful to select an appropriate distribution key because *it cannot be changed later*. Furthermore, any unique indexes (and therefore unique or primary keys) must be defined as a superset of the distribution key. That is, if a distribution key is defined, unique keys and primary keys must include all of the same columns as the distribution key (they might have more columns).

The size of a database partition of a table is the smaller amount of a specific limit associated with the type of table space and page size used, and the amount of disk space available. For example, assuming a large DMS table space with a 4 KB page size, the size of a table is the smaller amount of 8 TB multiplied by the number of database partitions and the amount of available disk space. See the related links for the complete list of database manager page size limits.

To create a table in a partitioned database environment using the command line, enter:

```
CREATE TABLE name>
  (<column_name> <data_type> <null_attribute>)
  IN <tagle_space_name>
  INDEX IN <index_space_name>
  LONG IN <long_space_name>
  DISTRIBUTE BY HASH (<column_name>)
```

Following is an example:

```
CREATE TABLE MIXREC (MIX_CNTL INTEGER NOT NULL,
                     MIX_DESC CHAR(20) NOT NULL,
                     MIX_CHR  CHAR(9) NOT NULL,
                     MIX_INT INTEGER NOT NULL,
```

```
MIX_INTS SMALLINT NOT NULL,
MIX_DEC DECIMAL NOT NULL,
MIX_FLT FLOAT NOT NULL,
MIX_DATE DATE NOT NULL,
MIX_TIME TIME NOT NULL,
MIX_TMSTMP TIMESTAMP NOT NULL)
IN MIXTS12
DISTRIBUTE BY HASH (MIX_INT)
```

In the preceding example, the table space is MIXTS12 and the distribution key is MIX_INT. If the distribution key is not specified explicitly, it is MIX_CNTL. (If no primary key is specified and no distribution key is defined, the distribution key is the first non-long column in the list.)

A row of a table, and all information about that row, always resides on the same database partition.

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.

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 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 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 block indexes and XML indexes are system generated indexes.

Example

Table A in Figure 25 on page 355 has an index based on the employee numbers in the table. This key value provides a pointer to the rows in the table. For example, employee number 19 points to employee KMP. An index allows efficient access to rows in a table by creating a path to the data through pointers.

Unique indexes can be created to ensure uniqueness of the index key. An *index key* is a column or an ordered collection of columns on which an index is defined. Using a unique index will ensure that the value of each index key in the indexed column or columns is unique.

Figure 25 on page 355 shows the relationship between an index and a table.

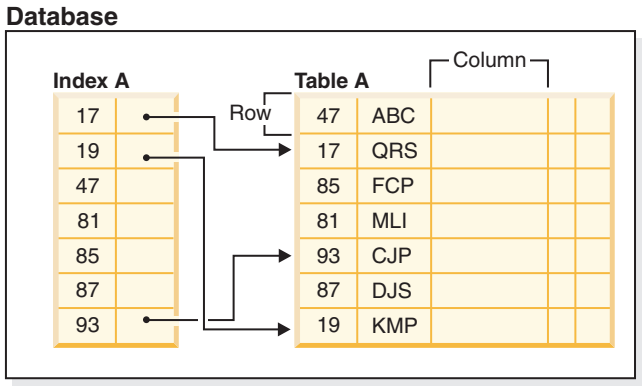


Figure 25. Relationship between an index and a table

Figure 26 illustrates the relationships among some database objects. It also shows that tables, indexes, and long data are stored in table spaces.

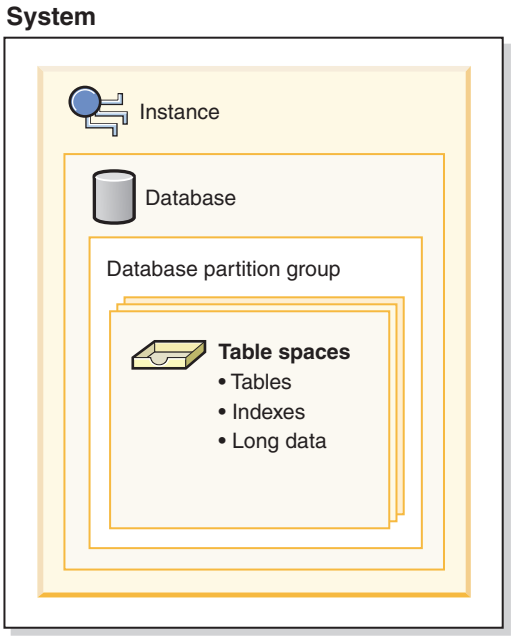


Figure 26. Relationships among selected database objects

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.

Note: When creating indexes, keep in mind that although they can improve read performance, they will negatively impact write performance. This is because for every row that the database manager writes to a table, it must also update any affected indexes. Therefore, you should create indexes only when there is a clear overall performance advantage.

When creating indexes, you must also take into account the structure of the tables and the type of queries that are most frequently performed on them. For example, columns appearing 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 be also be compressed. You can modify the indexes later, by enabling or disabling compression, 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 making 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 or not 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)` will be 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 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 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 will be used as the primary key index.

- If you have a *partitioned table*, by default, any index you create will be a *partitioned index* unless:
  - you are creating indexes over XML data
  - you are creating a unique index that does not include the partitioning key.

  You can also choose to have the index created as a *nonpartitioned index.*

  Partitioned indexes offer benefits when performing roll-in operations with partitioned tables (in other words, attaching a data partition to another table 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 SET INTEGRITY statement to perform index maintenance on the newly combined data partitions. Not only is this time consuming, it also can require a large amount of log space, depending on the number of rows 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 being indexed when planning the size of the database. Some of the indexing sizing considerations include:
  - Primary and unique key constraints will always create a system-generated unique index.
  - The creation of an MDC table will also create system-generated block indexes.
  - XML columns will always cause system-generated indexes to be created.
  - It is usually beneficial to create indexes on foreign key constraint columns.
  - Whether the index will be compressed or not (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 overhead, is *IndexPageSize* ÷ 4. The maximum 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.

  Note:
- During database upgrade, existing indexes will not be 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.

Creating indexes

Indexes can be created - among other reasons - 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, or 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.

About this task

This task assumes that you are creating an index on a nonpartitioned table.

To create an index using the CREATE INDEX statement from the command line, enter:

```
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.

Note: When a row is deleted or updated, the index keys are marked as deleted and are not physically removed from a page until clean up is done some time after the deletion or update is committed. These keys are referred to as pseudo-deleted keys. Such a clean up 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 using the CLEANUP ONLY ALL option of the REORG INDEXES utility.

Note: 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 will hang if a RAW device is used.

Configuring an instance to be Common Criteria compliant

A Common Criteria compliant DB2 instance must be configured so that:

- Any request made by a DB2 client to a DB2 server is authenticated by the server before being processed, and
- Communications occur using TCP/IP

The following topic provides the steps required to configure your environment so that it is Common Criteria compliant.

Configuring the DB2 database manager to be Common Criteria compliant

Immediately *after* installing the DB2 server, modify the values of the *authentication* and *svcename* database manager configuration parameters. Changing the values of these configuration parameters will ensure that your environment conforms to the Common Criteria requirements.

Before you begin

To update configuration parameter values, you require the SYSADM authority level.

About this task

1. Update the database manager configuration so that clients must authenticate themselves at the DB2 server via a user ID and password. Issue the following command:

   ```
   db2 update dbm cfg using authentication server
   ```

   For additional information about authentication types, see the topic “authentication - Authentication type” on page 1081

   Note: CLIENT is not a supported authentication type.
2. Update the database manager configuration to specify the TCP/IP port that DB2 database manager will use to await communications with remote client nodes. Issue the following command:

   ```
   db2 update dbm cfg using svcename port-number
   ```

   The port number that you specify must be the first of two consecutive ports that are reserved for use by database manager. For additional information about the *svcename* database manager configuration parameter, see the topic “svcename - TCP/IP service name” on page 1083
3. Stop the database manager. Issue the following command:

   ```
   db2stop
   ```
4. Start the database manager. Issue the following command:

   ```
   db2start
   ```

The updated database manager configuration parameters take effect when database manager is restarted.

Chapter 28. Altering a partitioned database environment

Altering a database partition group

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

About this task

Scaling your configuration

This chapter describes how you can manage database capacity, primarily by adding and dropping database partitions. Other methods of increasing capacity include adding CPUs and adding memory.

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, you should 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 an SMP system, processors share memory and storage system resources. All of the processors are in one system, so there are no additional overhead considerations such as communication between systems and coordination of tasks between systems. Utilities such as load, backup, 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 off-line.

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 intra-partition 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 will be 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.

Adding database partitions in partitioned database environments

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

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

- As an option on START DBM
- With the command-line processor ADD DBPARTITIONNUM command
- With the API function `sqleaddn`
- With the API function `sqlepstart`

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

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

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

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

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

You cannot change from a single-partition database to a multi-partition database by simply adding a database partition to your system. This is because the

redistribution of data across database partitions requires a distribution key on each affected table. The distribution keys are automatically generated when a table is created in a multi-partition database. In a single-partition database, distribution keys can be explicitly created with the CREATE TABLE or ALTER TABLE SQL statements.

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

Windows Considerations: If you are using Enterprise Server Edition on Windows and have no databases in the instance, use the db2ncrt command to scale the database system. If, however, you already have databases, use the START DBM ADD DBPARTITIONNUM command to ensure that a database partition is created for each existing database when you scale the system. On Windows, you should never manually edit the node configuration file (`db2nodes.cfg`), as this can introduce inconsistencies into the file.

Adding a database partition to a running database system

You can add new database partitions to a partitioned database environment while it is running and while applications are connected to databases. However, a newly added database partition does not become available to all databases until the database manager is shut down and restarted.

About this task

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

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

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

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

   When the START DBM command is complete, the new server is stopped.
2. Stop the database manager on all database partitions by running the STOP DBM command.

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

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

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

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

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

Adding a database partition to a stopped database system (Windows)

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

Before you begin

You must install the new server before you can create a database partition on it.

About this task

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

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

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

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

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

      When the START DBM command is complete, the new server is stopped.
   b. Stop the entire database manager by running the STOP DBM command.

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

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

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

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

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

Adding a database partition to a stopped database system (UNIX)

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

Before you begin

You must install the new server if it does not exist before you can create a database partition on it. In addition, your preparation should include the following tasks:

- Making executables accessible (using shared file-system mounts or local copies)
- Synchronizing operating system files with those on existing processors
- Ensuring that the `sqllib` directory is accessible as a shared file system
- Ensuring that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values

You must also register the host name with the name server or in the `.hosts` file in the `/etc` directory on all database partitions. The host name for the computer must be registered in `.rhosts` to run remote commands using rsh or rah.

About this task

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

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

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

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

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

      When the START DBM command is complete, the new server is stopped.
   b. Stop the entire database manager by running the STOP DBM command.

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

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

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

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

You can also update the configuration file manually, as follows:

1. Edit the db2nodes.cfg file and add the new database partition to it.

2. Issue the following command to start the new database partition: `START DBM DBPARTITIONNUM partitionnum`

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

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

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

Error recovery when adding database partitions

Adding database partitions does not fail as a result of nonexistent buffer pools because the database manager creates system buffer pools to provide default automatic support for all buffer pool page sizes. However, if one of these system buffer pools is used, performance might be seriously affected because the database manager created system buffer pools are very small. If a system buffer pool is used, a message is written to the administration notification log.

System buffer pools are used in database partition addition scenarios in the following circumstances:

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

  Consider the following examples:

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

     ```
     START DBM DBPARTITIONNUM 2 ADD DBPARTITIONNUM HOSTNAME newhost PORT 2
     ```
  2. You use the ADD DBPARTITIONNUM command after you manually update the `db2nodes.cfg` file with the new database partition description.

  One way to prevent these problems is to specify the WITHOUT TABLESPACES clause on the ADD DBPARTITIONNUM or the START DBM commands. After doing this, you need to use the CREATE BUFFERPOOL statement to create the buffer pools using the appropriate SIZE and PAGESIZE values, and associate the system temporary table spaces to the buffer pool using the ALTER TABLESPACE statement.
- You add database partitions to an existing database partition group that has one or more table spaces with a page size that is different from the default page size, which is 4 KB. This occurs because the non-default page-size buffer pools created on the new database partition have not been activated for the table spaces.

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

Consider the following example:

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

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

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

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

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

Dropping database partitions

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

Before you begin

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

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

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

About this task

To drop a database partition using the command line, issue the STOP DBM command with the DROP DBPARTITIONNUM parameter to drop the database partition. After the command completes successfully, the system is stopped. Then start the database manager with the START DBM command.

Redistributing data across database partitions

This chapter provides information about determining when to redistribute data across database partitions, how to perform the redistribution, and how to recover from redistribution errors.

Data redistribution

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

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

- REDISTRIBUTE DATABASE PARTITION GROUP command
- ADMIN_CMD system-defined procedure
- STEPWISE REDISTRIBUTE_DBPG system-defined procedure
- sqludrdt API

Data redistribution within a partitioned database is done for one of the following reasons:

- To rebalance data whenever a new database partition is added to the database environment or an existing database partition is removed.
- To introduce user specific data distribution across partitions.
- To secure sensitive data by isolating it within a particular partition.

Data redistribution is performed by connecting to a database at the catalog database partition and beginning a data redistribution operation for a specific partition group using one of the supported interfaces. Data redistribution relies on the existence of distribution key definitions for the tables within the partition group. The distribution key value for a row of data within the table is used to determine on which partition the row of data will be stored. A distribution key is generated automatically when a table is created in a multi-partition database partition group or can be explicitly defined using the CREATE TABLE or ALTER TABLE statements. By default during data redistribution, for each table within a specified nodegroup, table data is divided and redistributed evenly among the database partitions, however other distributions, such as a skewed distribution, can be achieved by specifying an input distribution map which defines how the data is to be distributed. Distribution maps can be generated during a data redistribution operation for future use or can be created manually.

Determining if data redistribution is needed

Determining the current data distribution for a database partition group or table can be helpful in determining if data redistribution is required and can be used to create a custom distribution map that can be used to specify how data should be distributed.

About this task

If a new database partition is added to a database partition group, or an existing database partition is dropped from a database partition group, then data redistribution should be performed in order to balance data among all the database partitions.

If no database partitions have been added or dropped from a database partition group, then data redistribution is usually only indicated when there is an unequal distribution of data among the database partitions of the database partition group. Please note that, in some cases, an unequal distribution of data can be desirable.

For example, if some database partitions reside on a particularly powerful machine, then it may be beneficial for those database partitions to contain larger volumes of data than other partitions.

To get information about the current distribution of data among database partitions in a database partition group, run the following query on the largest table (alternatively, a representative table) in the database partition group:

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

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

If the results of the query show that the distribution of data among database partitions is not as desired, then run the following query to get the distribution of data across hash partitions:

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

The output of this query can easily be used to construct the distribution file needed when the USING DISTFILE option of the REDISTRIBUTE DATABASE PARTITION GROUP command is specified (please refer to the Command Reference section for the REDISTRIBUTE DATABASE PARTITION GROUP command for a description of the format of the distribution file).

When the USING DISTFILE option is specified, the REDISTRIBUTE DATABASE PARTITION GROUP command will use the information in the file to generate a new partition map for the database partition group that results in a uniform distribution of data among database partitions.

If a uniform distribution is not desired, then the user can construct his or her own target partition map for the redistribution operation, which can be specified using the USING TARGETMAP option of the REDISTRIBUTE DATABASE PARTITION GROUP command.

Results

After doing this investigation you will know if your data is uniformly distributed or not or if data redistribution is required. If the data requires redistribution, you can plan to do this during a system maintenance opportunity using one of the supported interfaces.

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

Redistributing data can be successfully performed using the REDISTRIBUTE DATABASE PARTITION GROUP command. This is the recommended interface for performing data redistribution.

About this task

Restrictions

- See: Chapter 13, “Restrictions on data redistribution,” on page 53

To redistribute data across database partitions in a database partition group using the REDISTRIBUTE DATABASE PARTITION GROUP command:

1. Perform a backup of the database. See the BACKUP command.
2. Connect to the database partition that contains the system catalog tables. See the CONNECT command.
3. Issue the REDISTRIBUTE DATABASE PARTITION GROUP command.

   Note: In previous versions of the DB2 product, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

   Specify the following arguments:

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

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

   USING DISTFILE *distfile-name*
   OPTIONAL: Specifies that a customized distribution is desired and the file path name of a distribution file that contains data that defines the desired data skew. The contents of this file is used to generate a target distribution map.

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

   For details, refer to the REDISTRIBUTE DATABASE PARTITION GROUP command-line utility information.
4. Allow the command to run uninterrupted. When the command completes, if the data redistribution proceeded successfully:
   - Take a backup of all table spaces in the database partition group that are in the BACKUP PENDING state. Alternatively, a full database backup can be performed. NOTE: table spaces are only put into the BACKUP PENDING state if the database is recoverable and the NOT ROLLFORWARD RECOVERABLE option of the REDISTRIBUTE DATABASE PARTITION GROUP command is used.
   - Recreate any replicated materialized query tables dropped before redistribution.
   - If the STATISTICS NONE option of the REDISTRIBUTE DATABASE PARTITION GROUP command was specified or the NOT ROLLFORWARD RECOVERABLE option was omitted (both of which mean that the statistics were not collected during data redistribution) and there are tables in the database partition group possessing a statistics profile, execute the RUNSTATS command now to collect data distribution statistics for the SQL compiler and optimizer to use when it chooses data access plans for queries.
   - If the NOT ROLLFORWARD RECOVERABLE option was specified, delete the control files located in the following paths :
     – On Linux and UNIX operating systems: `DIAGPATH/redist/db_name/db_partitiongroup_name/timestamp/`

– On Windows operating systems: DIAGPATH\redist*db_name*\
*db_partitiongroup_name**timestamp*\

Results

Data redistribution should have completed successfully and information about the data redistribution process is available in the redistribution log file. Information about the distribution map that was used can be found in the DB2 explain tables.

Log space requirements for data redistribution

To successfully perform a data redistribution operation adequate log file space must be allocated before beginning the data redistribution operation to ensure that data redistribution is not interrupted.

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

When the REDISTRIBUTE DATABASE PARTITION GROUP command is used and the NOT ROLLFORWARD RECOVERABLE option is **not** used, or redistribution is perfomed from any other supported interface where the data redistribution is not rollforward recoverable:

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

Note: After you estimate the maximum amount of data to be inserted and deleted at a database partition, double that estimate to determine the peak size of the active log. If this estimate is greater than the active log limit of 1024 GB, then the data redistribution must be done in steps. Use the ″makepmap″ utility to generate

a series of target distribution maps, one for each step. You might also set the logsecond database configuration parameter to -1 to avoid most log space problems.

When the REDISTRIBUTE DATABASE PARTITION GROUP command is used and the NOT ROLLFORWARD RECOVERABLE option is used, or redistribution is performed from any other supported interface where the data redistribution is not rollforward recoverable:

- Log records are not created when rows are moved as part of data redistribution. This significantly reduces log file space requirements, however when this option is used when a rollforward recovery of the database is performed the redistribute operation log record cannot be rolled forward and any tables processed as part of the rollforward operation will remain in an UNAVAILABLE state. Please refer to the Command Reference for a discussion of the consequences of using the NOT ROLLFORWARD RECOVERABLE option.
- If the database partition group undergoing data redistribution does contain tables with long-field (LF) or large-object (LOB) data in the tables, the number of log records generated during data redistribution will be higher, because a log record is created for each row of data. In this case, expect the log space requirement per database partition to be roughly one third of the amount of data moving on that partition (i.e., data being sent, received or both). Regardless of the presence of LF/LOB data, on receiving partitions there is one type of log record that is written for which the number of such log records does depend on the amount of data moving: extent allocation log records. However, the total space required for these log records is small, and is never more than a tiny fraction of the total user data that is moving.

Redistribution event log file

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

When data redistribution is performed, information about each table which is processed is logged in a single redistribute event log file.

The event log file name is formatted like: `database-name.database-partition-group-name.timestamp.log`. The log files are located as follows:

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

The following is an example of an event log file name:

```
DB819.NG1.2007062419415651.log
```

This event log file is for a redistribute operation on a database named DB819 with a database partition group named NG1 that was created on June 24, 2007 at 7:41 PM local time.

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

- To provide general information about the redistribute operation, such as the old and new distribution maps.
- Provide users with information that will help them keep track of which tables have been redistributed so far by the utility.

- To provide information about each table that has been redistributed, including the indexing mode being used for the table, an indication of whether the table was successfully redistributed or not, and the starting and ending times for the redistribution operation on the table.

For more information about redistribute log file entries and how to recover from errors during data redistribution, see:

Redistributing database partition groups using the STEPWISE_REDISTRIBUTE_DBPG procedure

Redistributing data can be performed using the STEPWISE_REDISTRIBUTE_DBPG system-defined procedure.

Redistributing database partition groups can be done using the STEPWISE_REDISTRIBUTE_DBPG system-defined procedure and other system-defined procedures.

The following steps outline what must be done and an example that demonstrates these steps follows:

1. Analyze the database partition group regarding log space availability and data skew using *ANALYZE_LOG_SPACE procedure - Retrieve log space analysis information*.

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

   The generate_distfile function generates a data distribution file for the given table and saves it using the provided file name.
3. Create and report the content of a stepwise redistribution plan for the database partition group using *STEPWISE_REDISTRIBUTE_DBPG procedure - Redistribute part of a database partition group*.
4. Create a data distribution file for a given table using the *GET_SWRD_SETTINGS procedure - Retrieve redistribute information* and *SET_SWRD_SETTINGS procedure - Create or change redistribute registry*.

   The get_swrd_settings function reads the existing redistribute registry records for the given database partition group.

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

   The stepwise_redistribute_dbpg function redistributes part of the database partition group according to the input and the setting file.

Usage example

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

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

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

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

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

export DB2INSTANCE=$USER
export DB2COMM=TCPIP

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

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

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

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

# ------------------------------------------------------------------------------
# Write a step wise redistribution plan into a registry
#
# Setting the 10th parameter to 1, may cause a currently running step wise redistribute
# stored procedure to complete the current step and stop, until this parameter is reset
```

```
# to 0, and the redistribute stored procedure is called again.
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.set_swrd_settings('$dbpgName', 255, 0, ' ', '$distFile', 1000,
12, 2, 1, 0, '10,20,30', '50,50,50')"

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

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

Using Windows database partition servers

When working to change the characteristics of your configuration in a Windows environment, the tasks involved are carried out using specific utilities.

Listing database partition servers in an instance

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

About this task

The command is used as follows:

```
db2nlist
```

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

```
db2nlist /i:instName
```

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

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

```
db2nlist /s
```

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

Adding database partition servers to an instance (Windows)

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

About this task

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

The command has the following required parameters:

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

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

/u:username,password
The logon account name and password of the DB2 service.

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

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

- On every computer there must be a database partition server with a logical port 0.
- The port number cannot exceed the port range reserved for FCM communications in the services file in %SystemRoot%\system32\drivers\etc directory. For example, if you reserve a range of four ports for the current instance, then the maximum port number would be 3 (ports 1, 2, and 3; port 0 is for the default logical database partition). The port range is defined when db2icrt is used with the /r:base_port, end_port parameter.

There are also several optional parameters:

/g:network_name
Specifies the network name for the database partition server. If you do not specify this parameter, DB2 uses the first IP address it detects on your system.

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

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

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

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

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

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

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

Changing database partitions (Windows)

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

About this task

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

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

The command has the following required parameter:

```
db2nchg /n:node_number
```

The parameter /n: is the number of the database partition server's configuration you want to change. This parameter is required.

Optional parameters include:

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

/u:username,password
Changes the logon account name and password for the DB2 database service. If you do not specify this parameter, the logon account and password remain the same.

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

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

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

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

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

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

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

The DB2 database manager provides the capability of accessing DB2 database system registry variables at the instance level on a remote computer. Currently, DB2 database system registry variables are stored in three different levels: computer or global level, instance level, and database partition level. The registry variables stored at the instance level (including the database partition level) can be redirected to another computer by using DB2REMOTEPREG. When DB2REMOTEPREG is set, the DB2 database manager will access the DB2 database system registry variables from the computer pointed to by DB2REMOTEPREG. The db2set command would appear as:

```
db2set DB2REMOTEPREG=<remote workstation>
```

where <remote workstation> is the remote workstation name.

Note:

- Care should be taken in setting this option since all DB2 database instance profiles and instance listings will be located on the specified remote computer name.
- If your environment includes users from domains, ensure that the logon account associated with the DB2 instance service is a domain account. This ensures that the DB2 instance has the appropriate privileges to enumerate groups at the domain level.

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

Dropping a database partition from an instance (Windows)

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

About this task

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

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

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

The command has the following parameters:

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

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

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

Multiple logical partitions

When several database partition servers are running on the same computer, the computer is said to be running multiple logical partitions. This section describes when to use and how to configure multiple logical partitions.

Setting up multiple logical partitions

Typically, you configure DB2 Enterprise Server Edition to have one database partition server assigned to each computer. There are several situations, however, in which it would be advantageous to have several database partition servers running on the same computer.

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

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

- A standard configuration, where each computer has only one database partition server.
- A multiple logical partition configuration, where a computer has more than one database partition server.
- A configuration where several logical partitions run on each of several computers.

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

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

Configuring multiple logical partitions

There are two methods of configuring multiple logical partitions.

About this task

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

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

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

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

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

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

```
nodenumber hostname computername logical_port netname
```

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

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

Chapter 29. Concurrency, isolation levels, and locking

Concurrency, Isolation Levels, and Locking

Deadlocks

A deadlock is created when two applications lock data that is needed by the other, resulting in a situation in which neither application can continue executing.

For example, in Figure 27, there are two applications running concurrently: Application A and Application B. The first transaction for Application A is to update the first row in Table 1, and the second transaction is to update the second row in Table 2. Application B updates the second row in Table 2 first, and then the first row in Table 1. At time T1, Application A locks the first row in Table 1. At the same time, Application B locks the second row in Table 2. At time T2, Application A requests a lock on the second row in Table 2. However, at the same time, Application B tries to lock the first row in Table 1. Because Application A will not release its lock on the first row in Table 1 until it can complete an update to the second row in Table 2, and Application B will not release its lock on the second row in Table 2 until it can complete an update to the first row in Table 1, a deadlock occurs. The applications wait until one of them releases its lock on the data.

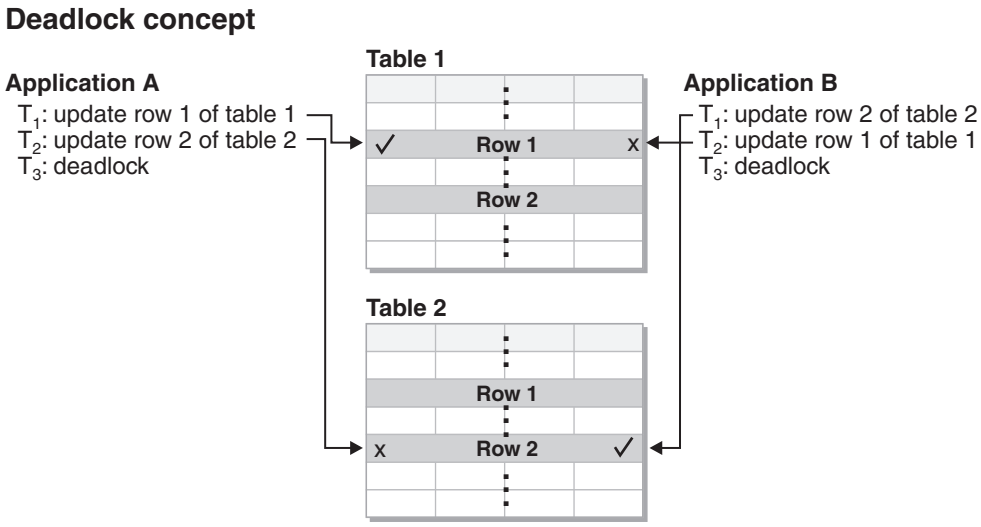


Figure 27. Deadlock between applications

Because applications do not voluntarily release locks on data that they need, a deadlock detector process is required to break deadlocks. The deadlock detector monitors information about agents that are waiting on locks, and awakens at intervals that are specified by the **dlchktime** database configuration parameter.

If it finds a deadlock, the deadlock detector arbitrarily selects one deadlocked process as the *victim process* to roll back. The victim process is awakened, and returns SQLCODE -911 (SQLSTATE 40001), with reason code 2, to the calling application. The database manager rolls back uncommitted transactions from the

selected process automatically. When the rollback operation is complete, locks that belonged to the victim process are released, and the other processes involved in the deadlock can continue.

To ensure good performance, select an appropriate value for **dlchktime**. An interval that is too short causes unnecessary overhead, and an interval that is too long allows deadlocks to linger.

In a partitioned database environment, the value of **dlchktime** is applied only at the catalog database partition. If a large number of deadlocks are occurring, increase the value of **dlchktime** to account for lock waits and communication waits.

To avoid deadlocks when applications read data that they intend to subsequently update:

- Use the FOR UPDATE clause when performing a select operation. This clause ensures that a U lock is set when a process attempts to read data, and it does not allow row blocking.
- Use the WITH RR or WITH RS and USE AND KEEP UPDATE LOCKS clauses in queries. These clauses ensure that a U lock is set when a process attempts to read data, and they allow row blocking.

In a federated system, the data that is requested by an application might not be available because of a deadlock at the data source. When this happens, the DB2 server relies on the deadlock handling facilities at the data source. If deadlocks occur across more than one data source, the DB2 server relies on data source timeout mechanisms to break the deadlocks.

To log more information about deadlocks, set the value of the **diaglevel** database manager configuration parameter to 4. The logged information includes the name of the locked object, the lock mode, and the application that is holding the lock. The current dynamic SQL and XQuery statement or static package name might also be logged.

Concurrency Control and Isolation Levels

Concurrency issues

Because many users access and change data in a relational database, the database manager must allow users to make these changes while ensuring that data integrity is preserved.

Concurrency refers to the sharing of resources by multiple interactive users or application programs at the same time. The database manager controls this access to prevent undesirable effects, such as:

- **Lost updates**. Two applications, A and B, might both read the same row and calculate new values for one of the columns based on the data that these applications read. If A updates the row and then B also updates the row, A's update lost.
- **Access to uncommitted data**. Application A might update a value, and B might read that value before it is committed. Then, if A backs out of that update, the calculations performed by B might be based on invalid data.
- **Non-repeatable reads**. Application A might read a row before processing other requests. In the meantime, B modifies or deletes the row and commits the

change. Later, if A attempts to read the original row again, it sees the modified row or discovers that the original row has been deleted.

- **Phantom reads**. Application A might execute a query that reads a set of rows based on some search criterion. Application B inserts new data or updates existing data that would satisfy application A's query. Application A executes its query again, within the same unit of work, and some additional ("phantom") values are returned.

Concurrency is not an issue for global temporary tables, because they are available only to the application that declares or creates them.

Concurrency control in federated database systems

A *federated database system* supports applications and users submitting SQL statements that reference two or more database management systems (DBMSs) in a single statement. To reference such data sources (each consisting of a DBMS and data), the DB2 server uses nicknames. *Nicknames* are aliases for objects in other DBMSs. In a federated system, the DB2 server relies on the concurrency control protocols of the database manager that hosts the requested data.

A DB2 federated system provides *location transparency* for database objects. For example, if information about tables and views is moved, references to that information (through nicknames) can be updated without changing the applications that request this information. When an application accesses data through nicknames, the DB2 server relies on concurrency control protocols at the data source to ensure that isolation levels are enforced. Although the DB2 server tries to match the isolation level that is requested at the data source with a logical equivalent, results can vary, depending on data source capabilities.

Isolation levels

The *isolation level* that is associated with an application process determines the degree to which the data that is being accessed by that process is locked or isolated from other concurrently executing processes. The isolation level is in effect for the duration of a unit of work.

The isolation level of an application process therefore specifies:

- The degree to which rows that are read or updated by the application are available to other concurrently executing application processes
- The degree to which the update activity of other concurrently executing application processes can affect the application

The isolation level for static SQL statements is specified as an attribute of a package and applies to the application processes that use that package. The isolation level is specified during the program preparation process by setting the ISOLATION bind or precompile option. For dynamic SQL statements, the default isolation level is the isolation level that was specified for the package preparing the statement. Use the SET CURRENT ISOLATION statement to specify a different isolation level for dynamic SQL statements that are issued within a session. For more information, see "CURRENT ISOLATION special register". For both static SQL statements and dynamic SQL statements, the *isolation-clause* in a *select-statement* overrides both the special register (if set) and the bind option value. For more information, see "Select-statement".

Isolation levels are enforced by locks, and the type of lock that is used limits or prevents access to the data by concurrent application processes. Declared temporary tables and their rows cannot be locked because they are only accessible to the application that declared them.

The database manager supports three general categories of locks:

Share (S)
Under an S lock, concurrent application processes are limited to read-only operations on the data.

Update (U)
Under a U lock, concurrent application processes are limited to read-only operations on the data, if these processes have not declared that they might update a row. The database manager assumes that the process currently looking at a row might update it.

Exclusive (X)
Under an X lock, concurrent application processes are prevented from accessing the data in any way. This does not apply to application processes with an isolation level of uncommitted read (UR), which can read but not modify the data.

Regardless of the isolation level, the database manager places exclusive locks on every row that is inserted, updated, or deleted. Thus, all isolation levels ensure that any row that is changed by an application process during a unit of work is not changed by any other application process until the unit of work is complete.

The database manager supports four isolation levels.

- "Repeatable read (RR)"
- "Read stability (RS)" on page 387
- "Cursor stability (CS)" on page 387
- "Uncommitted read (UR)" on page 388

Note: Some host database servers support the *no commit (NC)* isolation level. On other database servers, this isolation level behaves like the uncommitted read isolation level.

A detailed description of each isolation level follows, in decreasing order of performance impact, but in increasing order of the care that is required when accessing or updating data.

Repeatable read (RR)

The *repeatable read* isolation level locks all the rows that an application references during a unit of work (UOW). If an application issues a SELECT statement twice within the same unit of work, the same result is returned each time. Under RR, lost updates, access to uncommitted data, non-repeatable reads, and phantom reads are not possible.

Under RR, an application can retrieve and operate on the rows as many times as necessary until the UOW completes. However, no other application can update, delete, or insert a row that would affect the result set until the UOW completes. Applications running under the RR isolation level cannot see the uncommitted

changes of other applications. This isolation level ensures that all returned data remains unchanged until the time the application sees the data, even when temporary tables or row blocking is used.

Every referenced row is locked, not just the rows that are retrieved. For example, if you scan 10 000 rows and apply predicates to them, locks are held on all 10 000 rows, even if, say, only 10 rows qualify. Another application cannot insert or update a row that would be added to the list of rows referenced by a query if that query were to be executed again. This prevents phantom reads.

Because RR can acquire a considerable number of locks, this number might exceed limits specified by the **locklist** and **maxlocks** database configuration parameters. To avoid lock escalation, the optimizer might elect to acquire a single table-level lock for an index scan, if it appears that lock escalation is likely. If you do not want table-level locking, use the read stability isolation level.

While evaluating referential constraints, the DB2 server might occasionally upgrade the isolation level used on scans of the foreign table to RR, regardless of the isolation level that was previously set by the user. This results in additional locks being held until commit time, which increases the likelihood of a deadlock or a lock timeout. To avoid these problems, create an index that contains only the foreign key columns, and which the referential integrity scan can use instead.

Read stability (RS)

The *read stability* isolation level locks only those rows that an application retrieves during a unit of work. RS ensures that any qualifying row read during a UOW cannot be changed by other application processes until the UOW completes, and that any row changed by another application process cannot be read until the change is committed by that process. Under RS, access to uncommitted data and non-repeatable reads are not possible. However, phantom reads are possible.

This isolation level ensures that all returned data remains unchanged until the time the application sees the data, even when temporary tables or row blocking is used.

The RS isolation level provides both a high degree of concurrency and a stable view of the data. To that end, the optimizer ensures that table-level locks are not obtained until lock escalation occurs.

The RS isolation level is suitable for an application that:

- Operates in a concurrent environment
- Requires qualifying rows to remain stable for the duration of a unit of work
- Does not issue the same query more than once during a unit of work, or does not require the same result set when a query is issued more than once during a unit of work

Cursor stability (CS)

The *cursor stability* isolation level locks any row being accessed during a transaction while the cursor is positioned on that row. This lock remains in effect until the next row is fetched or the transaction terminates. However, if any data in the row was changed, the lock is held until the change is committed.

Under this isolation level, no other application can update or delete a row while an updatable cursor is positioned on that row. Under CS, access to the uncommitted data of other applications is not possible. However, non-repeatable reads and phantom reads are possible.

CS is the default isolation level. It is suitable when you want maximum concurrency and need to see only committed data.

Note: Under the *currently committed* semantics introduced in Version 9.7, only committed data is returned, as was the case previously, but now readers do not wait for updaters to release row locks. Instead, readers return data that is based on the currently committed version; that is, data prior to the start of the write operation.

Uncommitted read (UR)

The *uncommitted read* isolation level allows an application to access the uncommitted changes of other transactions. Moreover, UR does not prevent another application from accessing a row that is being read, unless that application is attempting to alter or drop the table.

Under UR, access to uncommitted data, non-repeatable reads, and phantom reads are possible. This isolation level is suitable if you run queries against read-only tables, or if you issue SELECT statements only, and seeing data that has not been committed by other applications is not a problem.

UR works differently for read-only and updatable cursors.

- Read-only cursors can access most of the uncommitted changes of other transactions.
- Tables, views, and indexes that are being created or dropped by other transactions are not available while the transaction is processing. Any other changes by other transactions can be read before they are committed or rolled back. Updatable cursors operating under UR behave as though the isolation level were CS.

If an uncommitted read application uses ambiguous cursors, it might use the CS isolation level when it runs. The ambiguous cursors can be escalated to CS if the value of the BLOCKING option on the PREP or BIND command is UNAMBIG (the default). To prevent this escalation:

- Modify the cursors in the application program to be unambiguous. Change the SELECT statements to include the FOR READ ONLY clause.
- Let the cursors in the application program remain ambiguous, but precompile the program or bind it with the BLOCKING ALL and STATICREADONLY YES options to enable the ambiguous cursors to be treated as read-only when the program runs.

Comparison of isolation levels

Table 85 summarizes the supported isolation levels.

Table 85. Comparison of isolation levels

	UR	CS	RS	RR
Can an application see uncommitted changes made by other application processes?	Yes	No	No	No

Table 85. Comparison of isolation levels (continued)

	UR	CS	RS	RR
Can an application update uncommitted changes made by other application processes?	No	No	No	No
Can the re-execution of a statement be affected by other application processes? [1]	Yes	Yes	Yes	No [2]
Can updated rows be updated by other application processes? [3]	No	No	No	No
Can updated rows be read by other application processes that are running at an isolation level other than UR?	No	No	No	No
Can updated rows be read by other application processes that are running at the UR isolation level?	Yes	Yes	Yes	Yes
Can accessed rows be updated by other application processes? [4]	Yes	Yes	No	No
Can accessed rows be read by other application processes?	Yes	Yes	Yes	Yes
Can the current row be updated or deleted by other application processes? [5]	Yes/No [6]	Yes/No [6]	No	No

Note:

1. An example of the *phantom read phenomenon* is as follows: Unit of work UW1 reads the set of *n* rows that satisfies some search condition. Unit of work UW2 inserts one or more rows that satisfy the same search condition and then commits. If UW1 subsequently repeats its read with the same search condition, it sees a different result set: the rows that were read originally plus the rows that were inserted by UW2.
2. If your label-based access control (LBAC) credentials change between reads, results for the second read might be different because you have access to different rows.
3. The isolation level offers no protection to the application if the application is both reading from and writing to a table. For example, an application opens a cursor on a table and then performs an insert, update, or delete operation on the same table. The application might see inconsistent data when more rows are fetched from the open cursor.
4. An example of the *non-repeatable read phenomenon* is as follows: Unit of work UW1 reads a row. Unit of work UW2 modifies that row and commits. If UW1 subsequently reads that row again, it might see a different value.
5. An example of the *dirty read phenomenon* is as follows: Unit of work UW1 modifies a row. Unit of work UW2 reads that row before UW1 commits. If UW1 subsequently rolls the changes back, UW2 has read nonexisting data.
6. Under UR or CS, if the cursor is not updatable, the current row can be updated or deleted by other application processes in some cases. For example, buffering might cause the current row at the client to be different from the current row at the server. Moreover, when using currently committed semantics under CS, a row that is being read might have uncommitted updates pending. In this case, the currently committed version of the row is always returned to the application.

Summary of isolation levels

Table 86 on page 390 lists the concurrency issues that are associated with different isolation levels.

Table 86. Summary of isolation levels

Isolation level	Access to uncommitted data	Non-repeatable reads	Phantom reads
Repeatable read (RR)	Not possible	Not possible	Not possible
Read stability (RS)	Not possible	Not possible	Possible
Cursor stability (CS)	Not possible	Possible	Possible
Uncommitted read (UR)	Possible	Possible	Possible

The isolation level affects not only the degree of isolation among applications but also the performance characteristics of an individual application, because the processing and memory resources that are required to obtain and free locks vary with the isolation level. The potential for deadlocks also varies with the isolation level. Table 87 provides a simple heuristic to help you choose an initial isolation level for your application.

Table 87. Guidelines for choosing an isolation level

Application type	High data stability required	High data stability not required
Read-write transactions	RS	CS
Read-only transactions	RR or RS	UR

Specifying the isolation level

Because the isolation level determines how data is isolated from other processes while the data is being accessed, you should select an isolation level that balances the requirements of concurrency and data integrity.

About this task

The isolation level that you specify is in effect for the duration of the unit of work (UOW). The following heuristics are used to determine which isolation level will be used when compiling an SQL or XQuery statement:

- For static SQL:
  - If an *isolation-clause* is specified in the statement, the value of that clause is used.
  - If an *isolation-clause* is not specified in the statement, the isolation level that was specified for the package when the package was bound to the database is used.
- For dynamic SQL:
  - If an *isolation-clause* is specified in the statement, the value of that clause is used.
  - If an *isolation-clause* is not specified in the statement, and a SET CURRENT ISOLATION statement has been issued within the current session, the value of the CURRENT ISOLATION special register is used.
  - If an *isolation-clause* is not specified in the statement, and a SET CURRENT ISOLATION statement has not been issued within the current session, the isolation level that was specified for the package when the package was bound to the database is used.
- For static or dynamic XQuery statements, the isolation level of the environment determines the isolation level that is used when the XQuery expression is evaluated.

Note: Many commercially-written applications provide a method for choosing the isolation level. Refer to the application documentation for information.

The isolation level can be specified in several different ways.

- **At the statement level:**

  Note: Isolation levels for XQuery statements cannot be specified at the statement level.

  Use the WITH clause. The WITH clause cannot be used on subqueries. The WITH UR option applies to read-only operations only. In other cases, the statement is automatically changed from UR to CS.

  This isolation level overrides the isolation level that is specified for the package in which the statement appears. You can specify an isolation level for the following SQL statements:

  – DECLARE CURSOR
  – Searched DELETE
  – INSERT
  – SELECT
  – SELECT INTO
  – Searched UPDATE

- **For dynamic SQL within the current session:**

  Use the SET CURRENT ISOLATION statement to set the isolation level for dynamic SQL issued within a session. Issuing this statement sets the CURRENT ISOLATION special register to a value that specifies the isolation level for any dynamic SQL statements that are issued within the current session. Once set, the CURRENT ISOLATION special register provides the isolation level for any subsequent dynamic SQL statement that is compiled within the session, regardless of which package issued the statement. This isolation level is in effect until the session ends or until the SET CURRENT ISOLATION...RESET statement is issued.

- **At precompile or bind time:**

  For an application written in a supported compiled language, use the ISOLATION option of the PREP or BIND commands. You can also use the sqlaprep or sqlabndx API to specify the isolation level.

  – If you create a bind file at precompile time, the isolation level is stored in the bind file. If you do not specify an isolation level at bind time, the default is the isolation level that was used during precompilation.
  – If you do not specify an isolation level, the default level of cursor stability (CS) is used.

  To determine the isolation level of a package, execute the following query:

```
select isolation from syscat.packages
  where pkgname = 'pkgname'
    and pkgschema = 'pkgschema'
```

  where *pkgname* is the unqualified name of the package and *pkgschema* is the schema name of the package. Both of these names must be specified in uppercase characters.

- **When working with JDBC or SQLJ at run time:**

  Note: JDBC and SQLJ are implemented with CLI on DB2 servers, which means that the `db2cli.ini` settings might affect what is written and run using JDBC and SQLJ.

To create a package (and specify its isolation level) in SQLJ, use the SQLJ profile customizer (db2sqljcustomize command).

- **From CLI or ODBC at run time:**

  Use the CHANGE ISOLATION LEVEL command. With DB2 Call-level Interface (CLI), you can change the isolation level as part of the CLI configuration. At run time, use the SQLSetConnectAttr function with the SQL_ATTR_TXN_ISOLATION attribute to set the transaction isolation level for the current connection referenced by the *ConnectionHandle* argument. You can also use the TXNISOLATION keyword in the `db2cli.ini` file.

- **On database servers that support REXX™:**

  When a database is created, multiple bind files that support the different isolation levels for SQL in REXX are bound to the database. Other command line processor (CLP) packages are also bound to the database when a database is created.

  REXX and the CLP connect to a database using the default CS isolation level. Changing this isolation level does not change the connection state.

  To determine the isolation level that is being used by a REXX application, check the value of the SQLISL predefined REXX variable. The value is updated each time that the CHANGE ISOLATION LEVEL command executes.

Results

Concurrency Control and Locking

Locks and concurrency control

To provide concurrency control and prevent uncontrolled data access, the database manager places locks on buffer pools, tables, data partitions, table blocks, or table rows.

A *lock* associates a database manager resource with an application, called the *lock owner*, to control how other applications access the same resource.

The database manager uses row-level locking or table-level locking, as appropriate, based on:

- The isolation level specified at precompile time or when an application is bound to the database. The isolation level can be one of the following:
  - Uncommitted read (UR)
  - Cursor stability (CS)
  - Read stability (RS)
  - Repeatable read (RR)

  The different isolation levels are used to control access to uncommitted data, prevent lost updates, allow non-repeatable reads of data, and prevent phantom reads. To minimize performance impact, use the minimum isolation level that satisfies your application needs.
- The access plan selected by the optimizer. Table scans, index scans, and other methods of data access each require different types of access to the data.
- The LOCKSIZE attribute for the table. The LOCKSIZE clause on the ALTER TABLE statement indicates the granularity of the locks that are used when the table is accessed. The choices are: ROW for row locks, TABLE for table locks, or BLOCKINSERT for block locks on multidimensional clustering (MDC) tables only. When the BLOCKINSERT clause is used on an MDC table, row-level locking is performed, except during an insert operation, when block-level

locking is done instead. Use the ALTER TABLE...LOCKSIZE BLOCKINSERT statement for MDC tables when transactions will be performing large inserts into disjointed cells. Use the ALTER TABLE...LOCKSIZE TABLE statement for read-only tables. This reduces the number of locks that are required for database activity. For partitioned tables, table locks are first acquired and then data partition locks are acquired, as dictated by the data that will be accessed.

- The amount of memory devoted to locking, which is controlled by the **locklist** database configuration parameter. If the lock list fills up, performance can degrade because of lock escalations and reduced concurrency among shared objects in the database. If lock escalations occur frequently, increase the value of **locklist**, **maxlocks**, or both. To reduce the number of locks that are held at one time, ensure that transactions commit frequently.

A buffer pool lock (exclusive) is set whenever a buffer pool is created, altered, or dropped. You might encounter this type of lock when collecting system monitoring data. The name of the lock is the identifier (ID) for the buffer pool itself.

In general, row-level locking is used unless one of the following is true:

- The isolation level is uncommitted read
- The isolation level is repeatable read and the access plan requires a scan with no index range predicates
- The table LOCKSIZE attribute is TABLE
- The lock list fills up, causing lock escalation
- An explicit table lock has been acquired through the LOCK TABLE statement, which prevents concurrent application processes from changing or using a table

In the case of an MDC table, block-level locking is used instead of row-level locking when:

- The table LOCKSIZE attribute is BLOCKINSERT
- The isolation level is repeatable read and the access plan involves predicates
- A searched update or delete operation involves predicates on dimension columns only

The duration of row locking varies with the isolation level being used:

- UR scans: No row locks are held unless row data is changing.
- CS scans: Row locks are generally held only while the cursor is positioned on the row. Note that in some cases, locks might not be held at all during a CS scan.
- RS scans: Qualifying row locks are held only for the duration of the transaction.
- RR scans: All row locks are held for the duration of the transaction.

Lock attributes

Database manager locks have several basic attributes.

These attributes include the following:

Mode The type of access allowed for the lock owner, as well as the type of access allowed for concurrent users of the locked object. It is sometimes referred to as the *state* of the lock.

Object
The resource being locked. The only type of object that you can lock explicitly is a table. The database manager also sets locks on other types of resources, such as rows and table spaces. Block locks can also be set for

multidimensional clustering (MDC) tables, and data partition locks can be set for partitioned tables. The object being locked determines the *granularity* of the lock.

Lock count
The length of time during which a lock is held. The isolation level under which a query runs affects the lock count.

Table 88 lists the lock modes and describes their effects, in order of increasing control over resources.

Table 88. Lock Mode Summary

Lock Mode	Applicable Object Type	Description
IN (Intent None)	Table spaces, blocks, tables, data partitions	The lock owner can read any data in the object, including uncommitted data, but cannot update any of it. Other concurrent applications can read or update the table.
IS (Intent Share)	Table spaces, blocks, tables, data partitions	The lock owner can read data in the locked table, but cannot update this data. Other applications can read or update the table.
IX (Intent Exclusive)	Table spaces, blocks, tables, data partitions	The lock owner and concurrent applications can read and update data. Other concurrent applications can both read and update the table.
NS (Scan Share)	Rows	The lock owner and all concurrent applications can read, but not update, the locked row. This lock is acquired on rows of a table, instead of an S lock, where the isolation level of the application is either RS or CS.
NW (Next Key Weak Exclusive)	Rows	When a row is inserted into an index, an NW lock is acquired on the next row. This occurs only if the next row is currently locked by an RR scan. The lock owner can read but not update the locked row. This lock mode is similar to an X lock, except that it is also compatible with NS locks.
S (Share)	Rows, blocks, tables, data partitions	The lock owner and all concurrent applications can read, but not update, the locked data.
SIX (Share with Intent Exclusive)	Tables, blocks, data partitions	The lock owner can read and update data. Other concurrent applications can read the table.
U (Update)	Rows, blocks, tables, data partitions	The lock owner can update data. Other units of work can read the data in the locked object, but cannot update it.
X (Exclusive)	Rows, blocks, tables, buffer pools, data partitions	The lock owner can both read and update data in the locked object. Only uncommitted read (UR) applications can access the locked object.
Z (Super Exclusive)	Table spaces, tables, data partitions	This lock is acquired on a table under certain conditions, such as when the table is altered or dropped, an index on the table is created or dropped, or for some types of table reorganization. No other concurrent application can read or update the table.

Lock granularity

If one application holds a lock on a database object, another application might not be able to access that object. For this reason, row-level locks, which minimize the amount of data that is locked and therefore inaccessible, are better for maximum concurrency than block-level, data partition-level, or table-level locks.

However, locks require storage and processing time, so a single table lock minimizes lock overhead.

The LOCKSIZE clause of the ALTER TABLE statement specifies the granularity of locks at the row, data partition, block, or table level. Row locks are used by default. Use of this option in the table definition does not prevent normal lock escalation from occurring.

The ALTER TABLE statement specifies locks globally, affecting all applications and users that access that table. Individual applications might use the LOCK TABLE statement to specify table locks at an application level instead.

A permanent table lock defined by the ALTER TABLE statement might be preferable to a single-transaction table lock using the LOCK TABLE statement if:

- The table is read-only, and will always need only S locks. Other users can also obtain S locks on the table.
- The table is usually accessed by read-only applications, but is sometimes accessed by a single user for brief maintenance, and that user requires an X lock. While the maintenance program is running, read-only applications are locked out, but in other circumstances, read-only applications can access the table concurrently with a minimum of locking overhead.

For a multidimensional clustering (MDC) table, you can specify BLOCKINSERT with the LOCKSIZE clause in order to use block-level locking during insert operations only. When BLOCKINSERT is specified, row-level locking is performed for all other operations, but only minimally for insert operations. That is, block-level locking is used during the insertion of rows, but row-level locking is used to lock the next key if repeatable read (RR) scans are encountered in the record ID (RID) indexes as they are being updated. BLOCKINSERT locking might be beneficial when:

- There are multiple transactions doing mass insertions into separate cells
- Concurrent insertions into the same cell by multiple transactions is not occurring, or it is occurring with enough data inserted per cell by each of the transactions that the user is not concerned that each transaction will insert into separate blocks

Preventing lock-related performance issues

There are guidelines to help you tune locking behavior for enhanced concurrency and data integrity.

- Create small units of work with frequent COMMIT statements to promote concurrent access of data by many users.

  Include COMMIT statements when your application is logically consistent; that is, when the data you have changed is consistent. When a COMMIT statement is issued, locks are released, except for table locks that are associated with cursors declared as WITH HOLD.
- Close a WITH HOLD cursor before issuing a COMMIT statement.

  In some situations, locks remain after the result set is closed and the transaction is committed. Closing a WITH HOLD cursor before issuing a COMMIT statement ensures that locks are released.
- Execute INSERT statements as separate units of work.

  In some situations, locks remain after the result set is closed and the transaction is committed. Executing INSERT statements as separate units of work ensures that locks are released.
- Specify an appropriate isolation level.

  Locks are acquired even if your application simply reads rows. Shared locks are acquired under the repeatable read, read stability, and cursor stability isolation

levels in read-only applications. Catalog locks are acquired even in uncommitted read applications that use dynamic SQL or XQuery statements. It is therefore important to commit read-only units of work. Under repeatable read and read stability, all locks are held until a COMMIT statement is issued. This prevents other processes from updating locked data, unless you close a cursor using the WITH RELEASE clause.

The database manager ensures that your application does not retrieve uncommitted data (rows that have been updated by other applications but that are not yet committed) unless you are using the uncommitted read isolation level.

- Use the LOCK TABLE statement appropriately.

  This statement locks an entire table. Parent and dependent tables of the specified table are not locked. You must determine whether locking other tables that can be accessed is necessary to achieve expected results in terms of concurrency and performance. The lock is not released until the unit of work is committed or rolled back.

  The IN SHARE MODE option prevents concurrent application processes from executing any but read-only operations on the table. The IN EXCLUSIVE MODE option prevents concurrent application processes from executing *any* operations on the table. EXCLUSIVE MODE does not prevent concurrent application processes that are running under the uncommitted read (UR) isolation level from executing read-only operations on the table.

- Use the ALTER TABLE statement.

  The ALTER TABLE statement with the LOCKSIZE option is an alternative to the LOCK TABLE statement. The LOCKSIZE option allows you to specify a lock granularity of row or table at next access. For MDC tables, it also allows you to specify a lock granularity of block for insert operations. None of the LOCKSIZE options prevent normal lock escalation.

  Table locks might improve query performance by limiting the number of locks that need to be acquired. However, concurrency might be reduced, because the entire table is locked. For MDC tables, block locks might improve the performance of insert operations by avoiding row locks. Row-level locking is still performed for all other operations, and is performed on key insertions to protect repeatable read (RR) scanners.

- Close cursors to release the locks that they hold.

  When you close a cursor with the CLOSE statement that includes the WITH RELEASE clause, the database manager attempts to release all read locks that have been held for that cursor.

  – Table-level read locks are: IS, S, and U
  – Row-level read locks are: S, NS, and U
  – Block-level read locks are: IS, S, and U

  The WITH RELEASE clause has no effect on cursors that are operating under the CS or UR isolation level. When specified for cursors that are operating under the RS or RR isolation level, the WITH RELEASE clause negates some of the guarantees of those isolation levels. Specifically, an RS cursor might experience the *non-repeatable read* phenomenon, and an RR cursor might experience either the non-repeatable read phenomenon or the *phantom read* phenomenon.

  If a cursor that originally operated under RR or RS is reopened after being closed WITH RELEASE, new read locks are acquired.

  In CLI applications, the DB2 CLI connection attribute SQL_ATTR_CLOSE_BEHAVIOR can be used to achieve the same results as CLOSE...WITH RELEASE.

- When you change the settings of configuration parameters that affect locking in a partitioned database environment, ensure that the changes are made at each database partition.

Correcting lock escalation problems

The database manager can automatically escalate locks from row or block level to table level.

About this task

For partitioned tables, the database manager can automatically escalate locks from row or block level to data partition level. The **maxlocks** database configuration parameter specifies when lock escalation is to be triggered. The table that acquires the lock that triggers lock escalation might not be affected. Locks are first escalated for the table with the most locks, beginning with tables for which large object (LOB) and long VARCHAR descriptors are locked. Then, locks are escalated for the table with the next highest number of locks, and so on, until the number of locks held is decreased to about half of the value specified by **maxlocks**.

In a well designed database, lock escalation is rare. If lock escalation reduces concurrency to an unacceptable level (as indicated by the **lock_escalation** monitor element or the **db.lock_escal_rate** health indicator) you should analyze the problem and decide on the best course of action.

First, ensure that lock escalation information is being recorded. Set the value of the **notifylevel** database manager configuration parameter to 3, which is the default, or to 4. At **notifylevel** 2, only error SQLCODEs are recorded. When lock escalation fails at **notifylevel** 3 or 4, information about the table for which lock escalation failed is recorded as well. At **notifylevel** 4, the query is also logged if it is a currently executing dynamic SQL statement.

Use the following steps to diagnose the cause of unacceptable lock escalations and to apply a remedy.

1. Obtain information from the administration notification log about all tables whose locks have been escalated. This log file includes the following information:
   - The number of locks currently held
   - The number of locks needed before lock escalation is completed
   - The table identifier and table name of each table being escalated
   - The number of non-table locks currently held
   - The new table-level lock to be acquired as part of the escalation. Usually, an S or X lock is acquired.
   - The internal return code that is associated with the acquisition of the new table-level lock
2. Use the information in the administration notification log to decide how to resolve the escalation problem. Consider the following options:
   - Increase the number of locks that are allowed globally by increasing the value of the **maxlocks** or **locklist** database configuration parameters, or both. In a partitioned database system, make this change on all database partitions.

     You might choose this method if concurrent access to the table by other processes is most important. However, the overhead of obtaining row-level locks can cause more delays to other processes than the time that is saved by increased concurrency.

- Adjust the processes that caused the escalation. For these processes, consider issuing LOCK TABLE statements explicitly.
- Change the degree of isolation. Note that this might lead to decreased concurrency.
- Increase the frequency of commits to reduce the number of locks that are held at any given time.
- Consider frequent commits for transactions that require LONG VARCHAR or large object (LOB) data. Although this kind of data is not retrieved from disk until the result set is materialized, the descriptor is locked when the data is first referenced. As a result, many more locks might be held than would be the case with regular data.

Lock type compatibility

Lock compatibility becomes an issue when one application holds a lock on an object and another application requests a lock on the same object. When the two lock modes are compatible, the request for a second lock on the object can be granted.

If the lock mode of the requested lock is not compatible with the lock that is already held, the lock request cannot be granted. Instead, the request must wait until the first application releases its lock, and all other existing incompatible locks are released.

Table 89 shows which lock types are compatible (indicated by a **yes**) and which types are not (indicated by a **no**). Note that a timeout can occur when a requestor is waiting for a lock.

Table 89. Lock Type Compatibility

	State of Held Resource										
State Being Requested	**None**	**IN**	**IS**	**NS**	**S**	**IX**	**SIX**	**U**	**X**	**Z**	**NW**
None	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
IN (Intent None)	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes
IS (Intent Share)	yes	yes	yes	yes	yes	yes	yes	yes	no	no	no
NS (Scan Share)	yes	yes	yes	yes	yes	no	no	yes	no	no	yes
S (Share)	yes	yes	yes	yes	yes	no	no	yes	no	no	no
IX (Intent Exclusive)	yes	yes	yes	no	no	yes	no	no	no	no	no
SIX (Share with Intent Exclusive)	yes	yes	yes	no	no	no	no	no	no	no	no
U (Update)	yes	yes	yes	yes	yes	no	no	no	no	no	no
X (Exclusive)	yes	yes	no	no	no	no	no	no	no	no	no
Z (Super Exclusive)	yes	no	no	no	no	no	no	no	no	no	no
NW (Next Key Weak Exclusive)	yes	yes	no	yes	no	no	no	no	no	no	no

Lock modes and access plans for standard tables

The type of lock that a standard table obtains depends on the isolation level that is in effect and on the data access plan that is being used.

The following tables show the types of locks that are obtained for standard tables under each isolation level for different access plans. Each entry has two parts: the table lock and the row lock. A hyphen indicates that a particular lock granularity is not available.

Tables 7-12 show the types of locks that are obtained when the reading of data pages is deferred to allow the list of rows to be further qualified using multiple indexes, or sorted for efficient prefetching.

- Table 1. Lock Modes for Table Scans with No Predicates
- Table 2. Lock Modes for Table Scans with Predicates
- Table 3. Lock Modes for RID Index Scans with No Predicates
- Table 4. Lock Modes for RID Index Scans with a Single Qualifying Row
- Table 5. Lock Modes for RID Index Scans with Start and Stop Predicates Only
- Table 6. Lock Modes for RID Index Scans with Index and Other Predicates (sargs, resids) Only
- Table 7. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with No Predicates
- Table 8. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with No Predicates
- Table 9. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Predicates (sargs, resids)
- Table 10. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Predicates (sargs, resids)
- Table 11. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Start and Stop Predicates Only
- Table 12. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Start and Stop Predicates Only

Note:

1. Block-level locks are also available for multidimensional clustering (MDC) tables.
2. Lock modes can be changed explicitly with the *lock-request-clause* of a SELECT statement.

Table 90. Lock Modes for Table Scans with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-	U/-	SIX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 91. Lock Modes for Table Scans with Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-	U/-	SIX/X	U/-	SIX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

Note: Under the UR isolation level, if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks are upgraded to an IS table lock or NS row locks.

Table 92. Lock Modes for RID Index Scans with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 93. Lock Modes for RID Index Scans with a Single Qualifying Row

Isolation level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/U	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 94. Lock Modes for RID Index Scans with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 95. Lock Modes for RID Index Scans with Index and Other Predicates (sargs, resids) Only

Isolation level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

Table 96. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S		X/-	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

Table 97. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 98. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S		IX/S	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

Table 99. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/-	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

Table 100. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S		IX/X	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

Table 101. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/-	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IS/-	IX/U	IX/X	IX/U	IX/X

Lock modes for MDC table and RID index scans

The type of lock that a multidimensional clustering (MDC) table obtains during a table or RID index scan depends on the isolation level that is in effect and on the data access plan that is being used.

The following tables show the types of locks that are obtained for MDC tables under each isolation level for different access plans. Each entry has three parts: the table lock, the block lock, and the row lock. A hyphen indicates that a particular lock granularity is not available.

Tables 9-14 show the types of locks that are obtained for RID index scans when the reading of data pages is deferred. Under the UR isolation level, if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks are upgraded to an IS table lock, an IS block lock, or NS row locks.

- Table 1. Lock Modes for Table Scans with No Predicates

- Table 2. Lock Modes for Table Scans with Predicates on Dimension Columns Only
- Table 3. Lock Modes for Table Scans with Other Predicates (sargs, resids)
- Table 4. Lock Modes for RID Index Scans with No Predicates
- Table 5. Lock Modes for RID Index Scans with a Single Qualifying Row
- Table 6. Lock Modes for RID Index Scans with Start and Stop Predicates Only
- Table 7. Lock Modes for RID Index Scans with Index Predicates Only
- Table 8. Lock Modes for RID Index Scans with Other Predicates (sargs, resids)
- Table 9. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with No Predicates
- Table 10. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with No Predicates
- Table 11. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Predicates (sargs, resids)
- Table 12. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Predicates (sargs, resids)
- Table 13. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Start and Stop Predicates Only
- Table 14. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Start and Stop Predicates Only

Note: Lock modes can be changed explicitly with the *lock-request-clause* of a SELECT statement.

Table 102. Lock Modes for Table Scans with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-/-	U/-/-	SIX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/U	IX/X/-	IX/I/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 103. Lock Modes for Table Scans with Predicates on Dimension Columns Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/X/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/U/-	X/X/-

Table 104. Lock Modes for Table Scans with Other Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/IX/X

Table 104. Lock Modes for Table Scans with Other Predicates (sargs, resids) (continued)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 105. Lock Modes for RID Index Scans with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

Table 106. Lock Modes for RID Index Scans with a Single Qualifying Row

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/IS/S	IX/IX/U	IX/IX/X	X/X/X	X/X/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

Table 107. Lock Modes for RID Index Scans with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/IS/S	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

Table 108. Lock Modes for RID Index Scans with Index Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 108. Lock Modes for RID Index Scans with Index Predicates Only (continued)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 109. Lock Modes for RID Index Scans with Other Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 110. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/S	IX/IX/S		X/-/-	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 111. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

Table 112. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/-	IX/IX/S		IX/IX/S	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 113. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 114. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/IS/S	IX/IX/S		IX/IX/X	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 115. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IS/-/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Lock modes for MDC block index scans

The type of lock that a multidimensional clustering (MDC) table obtains during a block index scan depends on the isolation level that is in effect and on the data access plan that is being used.

The following tables show the types of locks that are obtained for MDC tables under each isolation level for different access plans. Each entry has three parts: the table lock, the block lock, and the row lock. A hyphen indicates that a particular lock granularity is not available.

Tables 5-12 show the types of locks that are obtained for block index scans when the reading of data pages is deferred.

- Table 1. Lock Modes for Index Scans with No Predicates
- Table 2. Lock Modes for Index Scans with Predicates on Dimension Columns Only
- Table 3. Lock Modes for Index Scans with Start and Stop Predicates Only
- Table 4. Lock Modes for Index Scans with Predicates
- Table 5. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with No Predicates
- Table 6. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with No Predicates
- Table 7. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Predicates on Dimension Columns Only
- Table 8. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Predicates on Dimension Columns Only
- Table 9. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Start and Stop Predicates Only
- Table 10. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Start and Stop Predicates Only
- Table 11. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Other Predicates (sargs, resids)
- Table 12. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Other Predicates (sargs, resids)

Note: Lock modes can be changed explicitly with the *lock-request-clause* of a SELECT statement.

Table 116. Lock Modes for Index Scans with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/--/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 117. Lock Modes for Index Scans with Predicates on Dimension Columns Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 118. Lock Modes for Index Scans with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/-	IX/IX/S	IX/IX/S	IX/IX/S	IX/IX/S
RS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
CS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-

Table 119. Lock Modes for Index Scans with Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 120. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/--	IX/IX/S		X/--/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 121. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 122. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Predicates on Dimension Columns Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/--	IX/IX/--		IX/S/--	
RS	IS/IS/NS	IX/--/--		IX/--/--	
CS	IS/IS/NS	IX/--/--		IX/--/--	
UR	IN/IN/--	IX/--/--		IX/--/--	

Table 123. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Predicates on Dimension Columns Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/S/--	IX/X/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--

Table 124. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/--	IX/IX/--		IX/X/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 125. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/--	IX/IX/X		IX/X/--	
RS	IS/IS/NS	IN/IN/--		IN/IN/--	
CS	IS/IS/NS	IN/IN/--		IN/IN/--	
UR	IS/--/--	IN/IN/--		IN/IN/--	

Table 126. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Other Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/--	IX/IX/--		IX/IX/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 127. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Other Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Factors that affect locking

Several factors affect the mode and granularity of database manager locks.

These factors include:

- The type of processing that the application performs
- The data access method
- The values of various configuration parameters

Factors That Affect Locking

Locks and types of application processing

For the purpose of determining lock attributes, application processing can be classified as one of the following types: read-only, intent to change, change, and cursor controlled.

- Read-only

  This processing type includes all SELECT statements that are intrinsically read-only, have an explicit FOR READ ONLY clause, or are ambiguous, but the query compiler assumes that they are read-only because of the BLOCKING option value that the PREP or BIND command specifies. This type requires only share locks (IS, NS, or S).
- Intent to change

  This processing type includes all SELECT statements that have a FOR UPDATE clause, a USE AND KEEP UPDATE LOCKS clause, a USE AND KEEP EXCLUSIVE LOCKS clause, or are ambiguous, but the query compiler assumes that change is intended. This type uses share and update locks (S, U, or X for rows; IX, S, U, or X for blocks; and IX, U, or X for tables).
- Change

  This processing type includes UPDATE, INSERT, and DELETE statements, but not UPDATE WHERE CURRENT OF or DELETE WHERE CURRENT OF. This type requires exclusive locks (IX or X).
- Cursor controlled

  This processing type includes UPDATE WHERE CURRENT OF and DELETE WHERE CURRENT OF. This type requires exclusive locks (IX or X).

A statement that inserts, updates, or deletes data in a target table, based on the result from a subselect statement, does two types of processing. The rules for read-only processing determine the locks for the tables that return data in the subselect statement. The rules for change processing determine the locks for the target table.

Locks and data-access methods

An *access plan* is the method that the optimizer selects to retrieve data from a specific table. The access plan can have a significant effect on lock modes.

If an index scan is used to locate a specific row, the optimizer will usually choose row-level locking (IS) for the table. For example, if the EMPLOYEE table has an index on employee number (EMPNO), access through that index might be used to select information for a single employee:

```
select * from employee
  where empno = '000310'
```

If an index is not used, the entire table must be scanned in sequence to find the required rows, and the optimizer will likely choose a single table-level lock (S). For example, if there is no index on the column SEX, a table scan might be used to select all male employees, as follows:

```
select * from employee
  where sex = 'M'
```

Note: Cursor-controlled processing uses the lock mode of the underlying cursor until the application finds a row to update or delete. For this type of processing, no matter what the lock mode of the cursor might be, an exclusive lock is always obtained to perform the update or delete operation.

Locking in range-clustered tables works slightly differently from standard key locking. When accessing a range of rows in a range-clustered table, all rows in the range are locked, even when some of those rows are empty. In standard key locking, only rows with existing data are locked.

Deferred access to data pages implies that access to a row occurs in two steps, which results in more complex locking scenarios. The timing of lock acquisition and the persistence of locks depend on the isolation level. Because the repeatable read (RR) isolation level retains all locks until the end of a transaction, the locks acquired in the first step are held, and there is no need to acquire further locks during the second step. For the read stability (RS) and cursor stability (CS) isolation levels, locks must be acquired during the second step. To maximize concurrency, locks are not acquired during the first step, and the reapplication of all predicates ensures that only qualifying rows are returned.

Next-key locking

During insertion of a key into an index, the row that corresponds to the key that will follow the new key in the index is locked only if that row is currently locked by a repeatable read (RR) index scan.

The lock mode that is used for the next-key lock is NW (next key weak exclusive). This next-key lock is released before key insertion occurs; that is, before a row is inserted into the table.

Key insertion also occurs when updates to a row result in a change to the value of the index key for that row, because the original key value is marked deleted and the new key value is inserted into the index. For updates that affect only the include columns of an index, the key can be updated in place, and no next-key locking occurs.

During RR scans, the row that corresponds to the key that follows the end of the scan range is locked in S mode. If no keys follow the end of the scan range, an end-of-table lock is acquired to lock the end of the index. In the case of partitioned indexes for partitioned tables, locks are acquired to lock the end of each index partition, instead of just one lock for the end of the index. If the key that follows the end of the scan range is marked deleted, one of the following actions occurs:

- The scan continues to lock the corresponding rows until it finds a key that is not marked deleted
- The scan locks the corresponding row for that key
- The scan locks the end of the index

Evaluate uncommitted data through lock deferral

To improve concurrency, the database manager in some situations permits the deferral of row locks for CS or RS isolation scans until a row is known to satisfy the predicates of a query.

By default, when row-level locking is performed during a table or index scan, the database manager locks each scanned row whose commitment status is unknown before determining whether the row satisfies the predicates of the query.

To improve the concurrency of such scans, enable the **DB2_EVALUNCOMMITTED** registry variable so that predicate evaluation can occur on uncommitted data. A row that contains an uncommitted update might not satisfy the query, but if predicate evaluation is deferred until after the transaction completes, the row might indeed satisfy the query.

Uncommitted deleted rows are skipped during table scans, and the database manager skips deleted keys during index scans if the **DB2_SKIPDELETED** registry variable is enabled.

The **DB2_EVALUNCOMMITTED** registry variable setting applies at compile time for dynamic SQL or XQuery statements, and at bind time for static SQL or XQuery statements. This means that even if the registry variable is enabled at run time, the lock avoidance strategy is not deployed unless **DB2_EVALUNCOMMITTED** was enabled at bind time. If the registry variable is enabled at bind time but not enabled at run time, the lock avoidance strategy is still in effect. For static SQL or XQuery statements, if a package is rebound, the registry variable setting that is in effect at bind time is the setting that applies. An implicit rebind of static SQL or XQuery statements will use the current setting of the **DB2_EVALUNCOMMITTED** registry variable.

Applicability of evaluate uncommitted for different access plans

Table 128. RID Index Only Access

Predicates	Evaluate Uncommitted
None	No
SARGable	Yes

Table 129. Data Only Access (relational or deferred RID list)

Predicates	Evaluate Uncommitted
None	No
SARGable	Yes

Table 130. RID Index + Data Access

Predicates		Evaluate Uncommitted	
Index	**Data**	**Index access**	**Data access**
None	None	No	No
None	SARGable	No	No
SARGable	None	Yes	No
SARGable	SARGable	Yes	No

Table 131. Block Index + Data Access

Predicates		Evaluate Uncommitted	
Index	**Data**	**Index access**	**Data access**
None	None	No	No
None	SARGable	No	Yes
SARGable	None	Yes	No
SARGable	SARGable	Yes	Yes

Example

The following example provides a comparison between the default locking behavior and the evaluate uncommitted behavior. The table is the ORG table from the SAMPLE database.

```
DEPTNUMB DEPTNAME       MANAGER DIVISION   LOCATION
-------- -------------- ------- ---------- -------------
      10 Head Office        160 Corporate  New York
      15 New England         50 Eastern    Boston
      20 Mid Atlantic        10 Eastern    Washington
```

```
38 South Atlantic      30 Eastern    Atlanta
42 Great Lakes        100 Midwest    Chicago
51 Plains             140 Midwest    Dallas
66 Pacific            270 Western    San Francisco
84 Mountain           290 Western    Denver
```

The following transactions occur under the default cursor stability (CS) isolation level.

Table 132. Transactions against the ORG table under the CS isolation level

SESSION 1	SESSION 2
`connect to sample`	`connect to sample`
`+c update org set  deptnumb=5 where manager=160`	
	`select * from org where deptnumb >= 10`

The uncommitted UPDATE statement in Session 1 holds an exclusive lock on the first row in the table, preventing the query in Session 2 from returning a result set, even though the row being updated in Session 1 does not currently satisfy the query in Session 2. The CS isolation level specifies that any row that is accessed by a query must be locked while the cursor is positioned on that row. Session 2 cannot obtain a lock on the first row until Session 1 releases its lock.

Waiting for a lock in Session 2 can be avoided by using the evaluate uncommitted feature, which first evaluates the predicate and then locks the row. As such, the query in Session 2 would not attempt to lock the first row in the table, thereby increasing application concurrency. Note that this also means that predicate evaluation in Session 2 would occur with respect to the uncommitted value of `deptnumb=5` in Session 1. The query in Session 2 would omit the first row in its result set, despite the fact that a rollback of the update in Session 1 would satisfy the query in Session 2.

If the order of operations were reversed, concurrency could still be improved with the evaluate uncommitted feature. Under default locking behavior, Session 2 would first acquire a row lock prohibiting the searched UPDATE in Session 1 from executing, even though the Session 1 UPDATE statement would not change the row that is locked by the Session 2 query. If the searched UPDATE in Session 1 first attempted to examine rows and then locked them only if they qualified, the Session 1 query would be non-blocking.

Restrictions

- The **DB2_EVALUNCOMMITTED** registry variable must be enabled.
- The isolation level must be CS or RS.
- Row-level locking is in effect.
- SARGable evaluation predicates exist.
- Evaluate uncommitted is not applicable to scans on the system catalog tables.
- For multidimensional clustering (MDC) tables, block-level locking can be deferred for an index scan; however, block-level locking cannot be deferred for table scans.
- Lock deferral will not occur on a table that is executing an inplace table reorganization.
- For Iscan-Fetch plans, row-level locking is not deferred to the data access; rather, the row is locked during index access before moving to the row in the table.

- Deleted rows are unconditionally skipped during table scans, but deleted index keys are skipped only if the **DB2_SKIPDELETED** registry variable is enabled.

Option to disregard uncommitted insertions

The **DB2_SKIPINSERTED** registry variable controls whether or not uncommitted data insertions can be ignored for statements that use the cursor stability (CS) or the read stability (RS) isolation level.

Uncommitted insertions are handled in one of two ways, depending on the value of the **DB2_SKIPINSERTED** registry variable.

- When the value is ON, the DB2 server ignores uncommitted insertions, which in many cases can improve concurrency and is the preferred behavior for most applications. Uncommitted insertions are treated as though they had not yet occurred.
- When the value is OFF (the default), the DB2 server waits until the insert operation completes (commits or rolls back) and then processes the data accordingly. This is appropriate in certain cases. For example:
  - Suppose that two applications use a table to pass data between themselves, with the first application inserting data into the table and the second one reading it. The data must be processed by the second application in the order presented, such that if the next row to be read is being inserted by the first application, the second application must wait until the insert operation commits.
  - An application avoids UPDATE statements by deleting data and then inserting a new image of the data.

Table locking modes supported by the import utility

The import utility supports two table locking modes: offline, or ALLOW NO ACCESS, mode; and online, or ALLOW WRITE ACCESS mode.

ALLOW NO ACCESS mode prevents concurrent applications from accessing table data. ALLOW WRITE ACCESS mode allows concurrent applications both read and write access to the import target table. If no mode is explicitly specified, import runs in the default mode, ALLOW NO ACCESS. As well, the import utility is, by default, bound to the database with isolation level RS (read stability).

Offline import (ALLOW NO ACCESS)

In ALLOW NO ACCESS mode, import acquires an exclusive (X) lock on the target table is before inserting any rows. Holding a lock on a table has two implications:

- First, if there are other applications holding a table lock or row locks on the import target table, the import utility waits for those applications to commit or roll back their changes.
- Second, while import is running, any other application requesting locks waits for the import operation to complete.

Note: You can specify a locktimeout value, which prevents applications (including the import utility) from waiting indefinitely for a lock.
By requesting an exclusive lock at the beginning of the operation, import prevents deadlocks from occurring as a result of other applications working and holding row locks on the same target table.

Online import (ALLOW WRITE ACCESS)

In ALLOW WRITE ACCESS mode, the import utility acquires a nonexclusive (IX) lock on the target table. Holding this lock on the table has the following implications:

- If there are other applications holding an incompatible table lock, the import utility does not start inserting data until all of these applications commit or roll back their changes.
- While import is running, any other application requesting an incompatible table lock waits until the import commits or rolls back the current transaction. Note that import's table lock does not persist across a transaction boundary. As a result, online import has to request and potentially wait for a table lock after every commit.
- If there are other applications holding an incompatible row lock, the import utility stops inserting data until all of these applications commit or roll back their changes.
- While import is running, any other application requesting an incompatible row lock waits until the import operation commits or rolls back the current transaction.

To preserve the online properties, and to reduce the chance of a deadlock, an ALLOW WRITE ACCESS import periodically commits the current transaction and releases all row locks before escalating to an exclusive table lock. If you have not explicitly set a commit frequency, import performs commits as if COMMITCOUNT AUTOMATIC has been specified. No commits are performed if COMMITCOUNT is set to 0.

ALLOW WRITE ACCESS mode is not compatible with the following:

- Imports in REPLACE, CREATE, or REPLACE_CREATE mode
- Imports with buffered inserts
- Imports into a target view
- Imports into a hierarchy table
- Imports into a table with its lock granularity is set at the table level (set by using the LOCKSIZE parameter of the ALTER TABLE statement)

Chapter 30. DB2 commands

Command Line Processor (CLP)

db2 - Command line processor invocation

The db2 command starts the command line processor (CLP). The CLP is used to execute database utilities, SQL statements and online help. It offers a variety of command options, and can be started in:

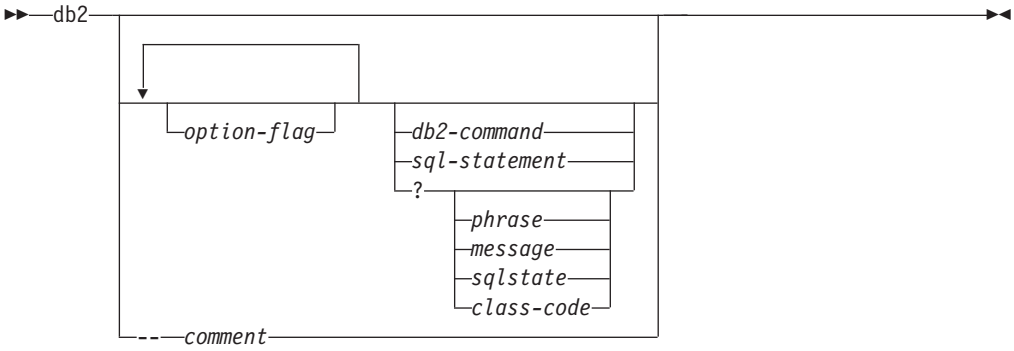
- Interactive input mode, characterized by the **db2 =>** input prompt
- Command mode, where each command must be prefixed by `db2`
- Batch mode, which uses the `-f` file input option.

On Windows operating systems, db2cmd opens the CLP-enabled DB2 window, and initializes the DB2 command line environment. Issuing this command is equivalent to clicking on the *DB2 Command Window* icon.

QUIT stops the command line processor. TERMINATE also stops the command line processor, but removes the associated back-end process and frees any memory that is being used. It is recommended that a TERMINATE be issued prior to every STOP DATABASE MANAGER (db2stop) command. It might also be necessary for a TERMINATE to be issued after database configuration parameters have been changed, in order for these changes to take effect. Existing connections should be reset before terminating the CLP.

The shell command (!), allows operating system commands to be executed from the interactive or the batch mode on UNIX based systems, and on Windows operating systems (`!ls` on UNIX, and `!dir` on Windows operating systems, for example).

Command Syntax



option-flag
Specifies a CLP option flag.

db2-command
Specifies a DB2 command.

sql-statement
Specifies an SQL statement.

? Requests CLP general help.

? phrase
Requests the help text associated with a specified command or topic. If the database manager cannot find the requested information, it displays the general help screen.

`? options` requests a description and the current settings of the CLP options. `? help` requests information about reading the online help syntax diagrams.

? message
Requests help for a message specified by a valid SQLCODE (`? sql10007n`, for example).

? sqlstate
Requests help for a message specified by a valid SQLSTATE.

? class-code
Requests help for a message specified by a valid class-code.

-- comment
Input that begins with the comment characters `--` is treated as a comment by the command line processor.

In each case, a blank space must separate the question mark (?) from the variable name.

Command line processor options

The CLP command options can be specified by setting the command line processor DB2OPTIONS environment variable (which must be in uppercase), or with command line flags.

Users can set options for an entire session using DB2OPTIONS.

View the current settings for the option flags and the value of DB2OPTIONS using LIST COMMAND OPTIONS. Change an option setting from the interactive input mode or a command file using UPDATE COMMAND OPTIONS.

The command line processor sets options in the following order:

1. Sets up default options.
2. Reads DB2OPTIONS to override the defaults.
3. Reads the command line to override DB2OPTIONS.
4. Accepts input from UPDATE COMMAND OPTIONS as a final interactive override.

Table 133 on page 419 summarizes the CLP option flags. These options can be specified in any sequence and combination. To turn an option on, prefix the corresponding option letter with a minus sign (-). To turn an option off, either prefix the option letter with a minus sign and follow the option letter with another minus sign, or prefix the option letter with a plus sign (+). For example, `-c` turns the auto-commit option on, and either `-c-` or `+c` turns it off. These option letters are not case sensitive, that is, `-a` and `-A` are equivalent.

Table 133. CLP Command Options

Option Flag	Description	Default Setting
-a	This option tells the command line processor to display SQLCA data.	OFF
-c	This option tells the command line processor to automatically commit SQL statements.	ON
-d	This option tells the command line processor to retrieve and display XML declarations of XML data.	OFF
-e{c | s}	This option tells the command line processor to display SQLCODE or SQLSTATE. These options are mutually exclusive.	OFF
-f *filename*	This option tells the command line processor to read command input from a file instead of from standard input.	OFF
-i	This option tells the command line processor to 'pretty print' the XML data with proper indentation. This option will only affect the result set of XQuery statements.	OFF
-l *filename*	This option tells the command line processor to log commands in a history file.	OFF
-m	This option tells the command line processor to print the number of rows affected for `INSERT/DELETE/UPDATE/MERGE`.	OFF
-n	Removes the new line character within a single delimited token. If this option is not specified, the new line character is replaced with a space. This option must be used with the `-t` option.	OFF
-o	This option tells the command line processor to display output data and messages to standard output.	ON
-p	This option tells the command line processor to display a command line processor prompt when in interactive input mode.	ON
-q	This option tells the command line processor to preserve whitespaces and linefeeds in strings delimited with single or double quotation marks. When option `q` is `ON`, option `n` is ignored.	OFF
-r *filename*	This option tells the command line processor to write the report generated by a command to a file.	OFF
-s	This option tells the command line processor to stop execution if errors occur while executing commands in a batch file or in interactive mode.	OFF
-t	This option tells the command line processor to use a semicolon (;) as the statement termination character.	OFF
-td*x* or -td*xx*	This option tells the command line processor to define and to use *x* or *xx* as the statement termination character or characters (1 or 2 characters in length).	OFF
-v	This option tells the command line processor to echo command text to standard output.	OFF
-w	This option tells the command line processor to display FETCH/SELECT warning messages.	ON

Table 133. CLP Command Options (continued)

Option Flag	Description	Default Setting
-x	This option tells the command line processor to return data without any headers, including column names. This flag will not affect all commands. It applies to SQL statements and some commands that are based on SQL statements such as LIST TABLES.	OFF
-z *filename*	This option tells the command line processor to redirect all output to a file. It is similar to the `-r` option, but includes any messages or error codes with the output.	OFF

Example

The AIX command:

```
export DB2OPTIONS='+a -c +ec -o -p'
```

sets the following default settings for the session:

```
Display SQLCA   - off
Auto Commit     -  on
Display SQLCODE - off
Display Output  -  on
Display Prompt  -  on
```

The following is a detailed description of these options:

Show SQLCA Data Option (-a):

Displays SQLCA data to standard output after executing a DB2 command or an SQL statement. The SQLCA data is displayed instead of an error or success message.

The default setting for this command option is `OFF` (`+a` or `-a-`).

The `-o` and the `-r` options affect the `-a` option; see the option descriptions for details.

Auto-commit Option (-c):

This option specifies whether each command or statement is to be treated independently. If set `ON` (`-c`), each command or statement is automatically committed or rolled back. If the command or statement is successful, it and all successful commands and statements that were issued before it with autocommit `OFF` (`+c` or `-c-`) are committed. If, however, the command or statement fails, it and all successful commands and statements that were issued before it with autocommit `OFF` are rolled back. If set `OFF` (`+c` or `-c-`), `COMMIT` or `ROLLBACK` must be issued explicitly, or one of these actions will occur when the next command with autocommit `ON` (`-c`) is issued.

The default setting for this command option is `ON`.

The auto-commit option does not affect any other command line processor option.

Example: Consider the following scenario:

1. `db2 create database test`
2. `db2 connect to test`
3. `db2 +c "create table a (c1 int)"`
4. `db2 select c2 from a`

The SQL statement in step 4 fails because there is no column named C2 in table A. Since that statement was issued with auto-commit ON (default), it rolls back not only the statement in step 4, but also the one in step 3, because the latter was issued with auto-commit OFF. The command:

```
db2 list tables
```

then returns an empty list.

XML Declaration Option (-d):

The -d option tells the command line processor whether to retrieve and display XML declarations of XML data.

If set ON (-d), the XML declarations will be retrieved and displayed. If set OFF (+d or -d-), the XML declarations will not be retrieved and displayed. The default setting for this command option is OFF.

The XML declaration option does not affect any other command line processor options.

Display SQLCODE/SQLSTATE Option (-e):

The -e{c|s} option tells the command line processor to display the SQLCODE (-ec) or the SQLSTATE (-es) to standard output. Options -ec and -es are not valid in CLP interactive mode.

The default setting for this command option is OFF (+e or -e-).

The -o and the -r options affect the -e option; see the option descriptions for details.

The display SQLCODE/SQLSTATE option does not affect any other command line processor option.

Example: To retrieve SQLCODE from the command line processor running on AIX, enter:

```
sqlcode=`db2 -ec +o db2—command`
```

Read from Input File Option (-f):

The -f *filename* option tells the command line processor to read input from a specified file, instead of from standard input. *Filename* is an absolute or relative file name which can include the directory path to the file. If the directory path is not specified, the current directory is used.

When the CLP is called with a file input option, it will automatically set the CLIENT APPLNAME special register to CLP *filename*.

When other options are combined with option -f, option -f must be specified last. For example:

```
db2 -tvf filename
```

This option cannot be changed from within the interactive mode.

The default setting for this command option is OFF (+f or -f-).

Commands are processed until the QUIT command or TERMINATE command is issued, or an end-of-file is encountered.

If both this option and a database command are specified, the command line processor does not process any commands, and an error message is returned.

Input file lines which begin with the comment characters `--` are treated as comments by the command line processor. Comment characters must be the first non-blank characters on a line.

Input file lines which begin with `(=` are treated as the beginning of a comment block. Lines which end with `=)` mark the end of a comment block. The block of input lines that begins at `(=` and ends at `=)` is treated as a continuous comment by the command line processor. Spaces before `(=` and after `=)` are allowed. Comments may be nested, and may be used nested in statements. The command termination character (`;`) cannot be used after `=)`.

If the `-f` *filename* option is specified, the `-p` option is ignored.

The read from input file option does not affect any other command line processor option.

Pretty Print Option (-i):

The `-i` option tells the command line processor to 'pretty print' the XML data with proper indentation. This option will only affect the result set of XQuery statements.

The default setting for this command option is `OFF` (`+i` or `-i-`).

The pretty print option does not affect any other command line processor options.

Log Commands in History File Option (-l):

The `-l` *filename* option tells the command line processor to log commands to a specified file. This history file contains records of the commands executed and their completion status. *Filename* is an absolute or relative file name which can include the directory path to the file. If the directory path is not specified, the current directory is used. If the specified file or default file already exists, the new log entry is appended to that file.

When other options are combined with option `-l`, option `-l` must be specified last. For example:

```
db2 -tvl filename
```

The default setting for this command option is `OFF` (`+l` or `-l-`).

The log commands in history file option does not affect any other command line processor option.

Display Number of Rows Affected Option (-m):

The `-m` option tells the command line processor whether or not to print the number of rows affected for INSERT, DELETE, UPDATE, or MERGE.

If set `ON (-m)`, the number of rows affected will be displayed for the statement of INSERT/DELETE/UPDATE/MERGE. If set `OFF (+m or -m-)`, the number of rows affected will not be displayed. For other statements, this option will be ignored. The default setting for this command option is `OFF`.

The `-o` and the `-r` options affect the `-m` option; see the option descriptions for details.

Remove New Line Character Option (-n):

Removes the new line character within a single delimited token. If this option is not specified, the new line character is replaced with a space. This option cannot be changed from within the interactive mode.

The default setting for this command option is OFF (+n or -n-).

This option must be used with the -t option; see the option description for details.

Display Output Option (-o):

The -o option tells the command line processor to send output data and messages to standard output.

The default setting for this command option is ON.

The interactive mode start-up information is not affected by this option. Output data consists of report output from the execution of the user-specified command, and SQLCA data (if requested).

The following options might be affected by the +o option:

- -r *filename*: Interactive mode start-up information is not saved.
- -e: SQLCODE or SQLSTATE is displayed on standard output even if +o is specified.
- -a: No effect if +o is specified. If -a, +o and -r*filename* are specified, SQLCA information is written to a file.

If both -o and -e options are specified, the data and either the SQLCODE or the SQLSTATE are displayed on the screen.

If both -o and -v options are specified, the data is displayed, and the text of each command issued is echoed to the screen.

The display output option does not affect any other command line processor option.

Display DB2 Interactive Prompt Option (-p):

The -p option tells the command line processor to display the command line processor prompt when the user is in interactive mode.

The default setting for this command option is ON.

Turning the prompt off is useful when commands are being piped to thecommand line processor . For example, a file containing CLP commands could be executed by issuing:

```
db2 +p < myfile.clp
```

The -p option is ignored if the -f *filename* option is specified.

The display DB2 interactive prompt option does not affect any other command line processor option.

Preserve Whitespaces and Linefeeds Option (-q):

The -q option tells the command line processor to preserve whitespaces and linefeeds in strings delimited with single or double quotation marks.

The default setting for this command option is OFF (+q or -q-).

If option -q is ON, option -n is ignored.

Save to Report File Option (-r):

The -r *filename* option causes any output data generated by a command to be written to a specified file, and is useful for capturing a report that would otherwise scroll off the screen. Messages or error codes are not written to the file. *Filename* is an absolute or relative file name which can include the directory path to the file. If the directory path is not specified, the current directory is used. New report entries are appended to the file.

The default setting for this command option is OFF (+r or -r-).

If the -a option is specified, SQLCA data is written to the file.

The -r option does not affect the -e option. If the -e option is specified, SQLCODE or SQLSTATE is written to standard output, not to a file.

If -r *filename* is set in DB2OPTIONS, the user can set the +r (or -r-) option from the command line to prevent output data for a particular command invocation from being written to the file.

The save to report file option does not affect any other command line processor option.

Stop Execution on Command Error Option (-s):

When commands are issued in interactive mode, or from an input file, and syntax or command errors occur, the -s option causes the command line processor to stop execution and to write error messages to standard output.

The default setting for this command option is OFF (+s or -s-). This setting causes the command line processor to display error messages, continue execution of the remaining commands, and to stop execution only if a system error occurs (return code 8).

The following table summarizes this behavior:

Table 134. CLP Return Codes and Command Execution

Return Code	-s Option Set	+s Option Set
0 (success)	execution continues	execution continues
1 (0 rows selected)	execution continues	execution continues
2 (warning)	execution continues	execution continues
4 (DB2 or SQL error)	execution stops	execution continues
8 (System error)	execution stops	execution stops

Statement Termination Character Options (-t and -td*x* or -td*xx*):

The -t option tells the command line processor to use a semicolon (;) as the statement termination character, and disables the backslash (\) line continuation character. This option cannot be changed from within the interactive mode.

The default setting for this command option is OFF (+t or -t-).

Note: If you use the CLP to issue XQuery statements, it is best to choose a termination character other than the semicolon. This ensures that statements or queries that use namespace declarations are not misinterpreted, since namespace declarations are also terminated by a semicolon.

To define termination characters 1 or 2 characters in length, use -td followed by the chosen character or characters. For example, -td%% sets %% as the statement termination characters. Alternatively, use the --#SET TERMINATOR directive in an input file to set the statement termination characters. For example:

```
db2 -td%% -f file1.txt
```

or

```
db2 -f file2.txt
```

where file2.txt contains the following as the first statement in the file:

```
--#SET TERMINATOR %%
```

The default setting for this command option is OFF.

The termination character or characters cannot be used to concatenate multiple statements from the command line, since checks for a termination symbol are performed on only the last one or two non-blank characters of each input line.

The statement termination character options do not affect any othercommand line processor option.

Verbose Output Option (-v):

The -v option causes the command line processor to echo (to standard output) the command text entered by the user prior to displaying the output, and any messages from that command. ECHO is exempt from this option.

The default setting for this command option is OFF (+v or -v-).

The -v option has no effect if +o (or -o-) is specified.

The verbose output option does not affect any other command line processor option.

Show Warning Messages Option (-w):

The -w option instructs the command line processor on whether or not to display warning messages that may occur during a query (FETCH/SELECT). Warnings can occur during various stages of the query execution which may result in the messages being displayed before, during or after the data is returned. To ensure the data returned does not contain warning message text this flag can be used.

The default setting for this command option is ON.

Suppress Printing of Column Headings Option (-x):

The -x option tells the command line processor to return data without any headers, including column names. This flag will not affect all commands. It applies to SQL statements and some commands that are based on SQL statements such as LIST TABLES.

The default setting for this command option is OFF.

Save all Output to File Option (-z):

The -z *filename* option causes all output generated by a command to be written to a specified file, and is useful for capturing a report that would otherwise scroll off the screen. It is similar to the -r option; in this case, however, messages, error codes, and other informational output are also written to the file. *Filename* is an absolute or relative file name which can include the directory path to the file. If the directory path is not specified, the current directory is used. New report entries are appended to the file.

The default setting for this command option is OFF (+z or -z-).

If the -a option is specified, SQLCA data is written to the file.

The -z option does not affect the -e option. If the -e option is specified, SQLCODE or SQLSTATE is written to standard output, not to a file.

If -z *filename* is set in DB2OPTIONS, the user can set the +z (or -z-) option from the command line to prevent output data for a particular command invocation from being written to the file.

The save all output to file option does not affect any other command line processor option.

Command line processor return codes

When the command line processor finishes processing a command or an SQL statement, it returns a return (or exit) code. These codes are transparent to users executing CLP functions from the command line, but they can be retrieved when those functions are executed from a shell script.

For example, the following Bourne shell script executes the GET DATABASE MANAGER CONFIGURATION command, then inspects the CLP return code:

```
db2 get database manager configuration
if [ "$?" = "0" ]
then echo "OK!"
fi
```

The return code can be one of the following:

Code	Description
0	DB2 command or SQL statement executed successfully
1	SELECT or FETCH statement returned no rows
2	DB2 command or SQL statement warning
4	DB2 command or SQL statement error
8	Command line processor system error

The command line processor does not provide a return code while a user is executing statements from interactive mode, or while input is being read from a file (using the `-f` option).

A return code is available only after the user quits interactive mode, or when processing of an input file ends. In these cases, the return code is the logical OR of the distinct codes returned from the individual commands or statements executed to that point.

For example, if a user in interactive mode issues commands resulting in return codes of 0, 1, and 2, a return code of 3 will be returned after the user quits interactive mode. The individual codes 0, 1, and 2 are not returned. Return code 3 tells the user that during interactive mode processing, one or more commands returned a 1, and one or more commands returned a 2.

A return code of 4 results from a negative SQLCODE returned by aDB2 command or an SQL statement. A return code of 8 results only if the command line processor encounters a system error.

If commands are issued from an input file or in interactive mode, and the command line processor experiences a system error (return code 8), command execution is halted immediately. If one or more DB2 commands or SQL statements end in error (return code 4), command execution stops if the `-s` (Stop Execution on Command Error) option is set; otherwise, execution continues.

Command line processor features

The command line processor operates as follows:

- The CLP command (in either case) is typed at the command prompt.
- The command is sent to the command shell by pressing the ENTER key.

- Output is automatically directed to the standard output device.
- Piping and redirection are supported.
- The user is notified of successful and unsuccessful completion.
- Following execution of the command, control returns to the operating system command prompt, and the user can enter more commands.
- When the CLP is called with a file input option, it will automatically set the CLIENT APPLNAME special register to `CLP` *`filename`*.

You can start the command line processor by either:

- typing the db2 command, or,
- on Linux operating systems, click **Main Menu** and, select **IBM DB2** → **Command Line Processor**.

Certain CLP commands and SQL statements require that the server instance is running and a database connection exists. Connect to a database by doing one of the following:

- Issue the SQL statement:

  ```
  db2 connect to database
  ```

- Establish an implicit connection to the default database defined by the DB2 registry variable **DB2DBDFT**.

If a command exceeds the character limit allowed at the command prompt, a backslash (\) can be used as the line continuation character. When the command line processor encounters the line continuation character, it reads the next line and concatenates the characters contained on both lines. Alternatively, the **-t** option can be used to set a different line termination character.

The command line processor recognizes a string called `NULL` as a null string. Fields that have been set previously to some value can later be set to NULL. For example,

```
db2 update database manager configuration using tm_database NULL
```

sets the **tm_database** field to NULL. This operation is case sensitive. A lowercase `null` is not interpreted as a null string, but rather as a string containing the letters `null`.

Customizing the Command Line Processor

It is possible to customize the interactive input prompt by using the **DB2_CLPPROMPT** registry variable. This registry variable can be set to any text string of maximum length 100 and can contain the tokens %i, %ia, %d, %da and %n. Specific values will be substituted for these tokens at run-time.

Table 135. DB2_CLPPROMPT tokens and run-time values

DB2_CLPPROMPT token	Value at run-time
%ia	Authorization ID of the current instance attachment
%i	Local alias of the currently attached instance. If no instance attachment exists, the value of the **DB2INSTANCE** registry variable. On Windows platforms only, if the **DB2INSTANCE** registry variable is not set, the value of the **DB2INSTDEF** registry variable.
%da	Authorization ID of the current database connection

Table 135. DB2_CLPPROMPT tokens and run-time values (continued)

DB2_CLPPROMPT token	Value at run-time
%d	Local alias of the currently connected database. If no database connection exists, the value of the **DB2DBDFT** registry variable.
%n	New line

- If any token has no associated value at runtime, the empty string is substituted for that token.
- The interactive input prompt will always present the authorization IDs, database names, and instance names in uppercase, so as to be consistent with the connection and attachment information displayed at the prompt.
- If the **DB2_CLPPROMPT** registry variable is changed within CLP interactive mode, the new value of **DB2_CLPPROMPT** will not take effect until CLP interactive mode has been closed and reopened.

You can specify the number of commands to be stored in the command history by using the **DB2_CLPHISTSIZE** registry variable. The HISTORY command lets you access the contents of the command history that you run within a CLP interactive mode session.

You can also specify the editor that is opened when you issue the EDIT command by using the **DB2_CLP_EDITOR** registry variable. From a CLP interactive session, the EDIT command opens an editor preloaded with a user-specified command which can then be edited and run.

Examples

If **DB2_CLPPROMPT** is defined as (%ia@%i, %da@%d), the input prompt will have the following values:

- No instance attachment and no database connection. **DB2INSTANCE** set to DB2. **DB2DBDFT** is not set.
  ```
  (@DB2, @)
  ```
- (Windows) No instance attachment and no database connection. **DB2INSTANCE** and **DB2DBDFT** not set. **DB2INSTDEF** set to DB2.
  ```
  (@DB2, @)
  ```
- No instance attachment and no database connection. **DB2INSTANCE** set to DB2. **DB2DBDFT** set to ″SAMPLE″.
  ```
  (@DB2, @SAMPLE)
  ```
- Instance attachment to instance ″DB2″ with authorization ID ″keon14″. **DB2INSTANCE** set to DB2. **DB2DBDFT** set to ″SAMPLE″.
  ```
  (KEON14@DB2, @SAMPLE)
  ```
- Database connection to database ″sample″ with authorization ID ″horton7″. **DB2INSTANCE** set to DB2. **DB2DBDFT** set to SAMPLE.
  ```
  (@DB2, HORTON7@SAMPLE)
  ```
- Instance attachment to instance ″DB2″ with authorization ID ″keon14″. Database connection to database ″sample″ with authorization ID ″horton7″. **DB2INSTANCE** set to DB2. **DB2DBDFT** not set.
  ```
  (KEON14@DB2, HORTON7@SAMPLE)
  ```

Using the Command Line Processor in command files

CLP requests to the database manager can be imbedded in a shell script command file. The following example shows how to enter the CREATE TABLE statement in a shell script command file:

```
db2 "create table mytable (name VARCHAR(20), color CHAR(10))"
```

For more information about commands and command files, see the appropriate operating system manual.

Command Line Processor design

The command line processor consists of two processes: the front-end process (the DB2 command), which acts as the user interface, and the back-end process (db2bp), which maintains a database connection.

Maintaining database connections

Each time that db2 is invoked, a new front-end process is started. The back-end process is started by the first db2 invocation, and can be explicitly terminated with TERMINATE. All front-end processes with the same parent are serviced by a single back-end process, and therefore share a single database connection.

For example, the following db2 calls from the same operating system command prompt result in separate front-end processes sharing a single back-end process, which holds a database connection throughout:

- `db2 'connect to sample'`,
- `db2 'select * from org'`,
- `. foo` (where `foo` is a shell script containing DB2 commands), and
- `db2 -tf myfile.clp`

The following invocations from the same operating system prompt result in separate database connections because each has a distinct parent process, and therefore a distinct back-end process:

- `foo`
- `. foo &`
- `foo &`
- `sh foo`

Communication between front-end and back-end processes

The front-end process and back-end processes communicate through three message queues: a request queue, an input queue, and an output queue.

Environment variables

The following environment variables offer a means of configuring communication between the two processes:

Table 136. Environment Variables

Variable	Minimum	Maximum	Default
DB2BQTIME	1 second	5294967295	1 second
DB2BQTRY	0 tries	5294967295	60 tries

Table 136. Environment Variables (continued)

Variable	Minimum	Maximum	Default
DB2RQTIME	1 second	5294967295	5 seconds
DB2IQTIME	1 second	5294967295	5 seconds

DB2BQTIME
When the command line processor is invoked, the front-end process checks if the back-end process is already active. If it is active, the front-end process reestablishes a connection to it. If it is not active, the front-end process activates it. The front-end process then idles for the duration specified by the **DB2BQTIME** variable, and checks again. The front-end process continues to check for the number of times specified by the **DB2BQTRY** variable, after which, if the back-end process is still not active, it times out and returns an error message.

DB2BQTRY
Works in conjunction with the **DB2BQTIME** variable, and specifies the number of times the front-end process tries to determine whether the back-end process is active.

The values of **DB2BQTIME** and **DB2BQTRY** can be increased during peak periods to optimize query time.

DB2RQTIME
Once the back-end process has been started, it waits on its request queue for a request from the front-end. It also waits on the request queue between requests initiated from the command prompt.

The **DB2RQTIME** variable specifies the length of time the back-end process waits for a request from the front-end process. At the end of this time, if no request is present on the request queue, the back-end process checks whether the parent of the front-end process still exists, and terminates itself if it does not exist. Otherwise, it continues to wait on the request queue.

DB2IQTIME
When the back-end process receives a request from the front-end process, it sends an acknowledgment to the front-end process indicating that it is ready to receive input via the input queue. The back-end process then waits on its input queue. It also waits on the input queue while a batch file (specified with the **-f** option) is executing, and while the user is in interactive mode.

The **DB2IQTIME** variable specifies the length of time the back-end process waits on the input queue for the front-end process to pass the commands. After this time has elapsed, the back-end process checks whether the front-end process is active, and returns to wait on the request queue if the front-end process no longer exists. Otherwise, the back-end process continues to wait for input from the front-end process.

To view the values of these environment variables, use LIST COMMAND OPTIONS.

The back-end environment variables inherit the values set by the front-end process at the time the back-end process is initiated. However, if the front-end environment

variables are changed, the back-end process will not inherit these changes. The back-end process must first be terminated, and then restarted (by issuing the db2 command) to inherit the changed values.

An example of when the back-end process must be terminated is provided by the following scenario:

1. User A logs on, issues some CLP commands, and then logs off without issuing TERMINATE.
2. User B logs on using the same window.
3. When user B issues certain CLP commands, they fail with message DB21016 (system error).

The back-end process started by user A is still active when user B starts using the CLP, because the parent of user B's front-end process (the operating system window from which the commands are issued) is still active. The back-end process attempts to service the new commands issued by user B; however, user B's front-end process does not have enough authority to use the message queues of the back-end process, because it needs the authority of user A, who created that back-end process. A CLP session must end with a TERMINATE command before a user starts a new CLP session using the same operating system window. This creates a fresh back-end process for each new user, preventing authority problems, and setting the correct values of environment variables (such as **DB2INSTANCE**) in the new user's back-end process.

CLP usage notes

Commands can be entered either in uppercase or in lowercase from the command prompt. However, parameters that are case sensitive to DB2 must be entered in the exact case desired. For example, the *comment-string* in the **WITH** clause of the CHANGE DATABASE COMMENT command is a case sensitive parameter.

Delimited identifiers are allowed in SQL statements.

Special characters, or metacharacters (such as $ & * ( ) ; < > ? \ ' ") are allowed within CLP commands. If they are used outside the CLP interactive mode, or the CLP batch input mode, these characters are interpreted by the operating system shell. Quotation marks or an escape character are required if the shell is not to take any special action.

For example, when executed inside an AIX Korn shell environment,

```
db2 select * from org where division > 'Eastern'
```

is interpreted as "select <the names of all files> from org where division". The result, an SQL syntax error, is redirected to the file `Eastern`. The following syntax produces the correct output:

```
db2 "select * from org where division > 'Eastern'"
```

Special characters vary from platform to platform. In the AIX Korn shell, the above example could be rewritten using an escape character (\), such as *, \>, or \'.

Most operating system environments allow input and output to be redirected. For example, if a connection to the SAMPLE database has been made, the following request queries the STAFF table, and sends the output to a file named `staflist.txt` in the `mydata` directory:

```
db2 "select * from staff" > mydata/staflist.txt
```

For environments where output redirection is not supported, CLP options can be used. For example, the request can be rewritten as

```
db2 -r mydata\staflist.txt "select * from staff"

db2 -z mydata\staflist.txt "select * from staff"
```

The command line processor is not a programming language. For example, it does not support host variables, and the statement,

```
db2 connect to :HostVar in share mode
```

is syntactically incorrect, because `:HostVar` is not a valid database name.

The command line processor represents SQL NULL values as hyphens (-). If the column is numeric, the hyphen is placed at the right of the column. If the column is not numeric, the hyphen is at the left.

To correctly display the national characters for single byte (SBCS) languages from the DB2 command line processor window, a True Type font must be selected. For example, in a Windows environment, open the command window properties notebook and select a font such as Lucinda Console.

The command line processor does not support national language support (NLS) characters in file path names. This particularly affects commands such as IMPORT, EXPORT, and REGISTER XMLSCHEMA, where problematic file path names would most frequently be encountered.

Security considerations for utilities

Privileges and authorities required to use the export utility

Privileges enable you to create, update, delete, or access database resources. Authority levels provide a method of mapping privileges to higher-level database manager maintenance and utility operations.

Together, privileges and authorities control access to the database manager and its database objects. You can access only those objects for which you have the appropriate authorization: that is, the required privilege or authority.

You must have DATAACCESS authority or the CONTROL or SELECT privilege for each table or view participating in the export operation.

When you are exporting LBAC-protected data, the session authorization ID must be allowed to read the rows or columns that you are trying to export. Protected rows that the session authorization ID is not authorized to read are not exported. If the SELECT statement includes any protected columns that the session authorization ID is not allowed to read, the export utility fails, and an error (SQLSTATE 42512) is returned.

Privileges, authorities, and authorization required to use backup

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the

database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the backup utility.

Privileges, authorities, and authorization required to use recover

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the recover utility.

Privileges, authorities, and authorization required to use restore

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to restore to an *existing* database from a full database backup. To restore to a *new* database, you must have SYSADM or SYSCTRL authority.

Authorization required for rollforward

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the rollforward utility.

Privileges and authorities required to use load

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

To load data into a table, you must have one of the following:

- DATAACCESS authority

- LOAD or DBADM authority on the database and
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.
  - SELECT privilege on SYSCAT.TABLES is required in some cases where LOAD queries the catalog tables.

Since all load processes (and all DB2 server processes, in general), are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

If the REPLACE option is specified, the session authorization ID must have the authority to drop the table.

On Windows, and Windows.NET operating systems where DB2 is running as a Windows service, if you are loading data from files that reside on a network drive, you must configure the DB2 service to run under a user account that has read access to these files.

Note:

- To load data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table.
- To load data into a table that has protected rows, the session authorization ID must have been granted a security label for write access that is part of the security policy protecting the table.

Issuing commands to multiple database partitions

This section describes how to issue commands to multiple database partitions, including problem resolution.

Issuing commands in partitioned database environments

In a partitioned database environment, you might want to issue commands to be run on computers in the instance, or on database partition servers (nodes). You can do so using the rah command or the db2_all command. The rah command allows you to issue commands that you want to run at computers in the instance.

If you want the commands to run at database partition servers in the instance, you run the db2_all command. This section provides an overview of these commands. The information that follows applies to partitioned database environments only.

On Windows, to run the rah command or the db2_all command, you must be logged on with a user account that is a member of the Administrators group.

On Linux and UNIX platforms, your login shell can be a Korn shell or any other shell; however, there are differences in the way the different shells handle commands containing special characters.

Also, on Linux and UNIX platforms, rah uses the remote shell program specified by the **DB2RSHCMD** registry variable. You can select between the two remote shell programs: ssh (for additional security), or rsh (or remsh for HP-UX). If **DB2RSHCMD** is not set, rsh (or remsh for HP-UX) is used. The ssh remote shell program is used to prevent the transmission of passwords in clear text in UNIX operating system environments. For a detailed description of how to configure one version of ssh for use with a DB2 DPF instance, see the following article: "Configure DB2 Universal Database for UNIX to use OpenSSH."

To determine the scope of a command, refer to the *Command Reference*, which indicates whether a command runs on a single database partition server, or on all of them. If the command runs on one database partition server and you want it to run on all of them, use db2_all. The exception is the db2trc command, which runs on all the logical nodes (database partition servers) on a computer. If you want to run db2trc on all logical nodes on all computers, use rah.

rah and db2_all commands overview

You can run the commands sequentially at one database partition server after another, or you can run the commands in parallel. On Linux and UNIX platforms, if you run the commands in parallel, you can either choose to have the output sent to a buffer and collected for display (the default behavior) or the output can be displayed at the computer where the command is issued.

On Windows, if you run the commands in parallel, the output is displayed at the computer where the command is issued.

To use the rah command, type:

```
rah command
```

To use the db2_all command, type:

```
db2_all command
```

To obtain help about rah syntax, type

```
rah "?"
```

The command can be almost anything which you could type at an interactive prompt, including, for example, multiple commands to be run in sequence. On Linux and UNIX platforms, you separate multiple commands using a semicolon (;). On Windows, you separate multiple commands using an ampersand (&). Do not use the separator character following the last command.

The following example shows how to use the db2_all command to change the database configuration on all database partitions that are specified in the node configuration file. Because the ; character is placed inside double quotation marks, the request will run concurrently:

```
db2_all ";DB2 GET DB CFG FOR sample USING LOGFILSIZ 100"
```

rah and db2_all commands

This topic includes descriptions of the rah and db2_all commands.

Command
Description

rah Runs the command on all computers.

db2_all
Runs the command on all database partition servers that you specify.

db2_kill
Abruptly stops all processes being run on multiple database partition servers and cleans up all resources on all database partition servers. This command renders your databases inconsistent. Do *not* issue this command except under direction from IBM Software Support or as directed to recover from a sustained trap.

db2_call_stack
On Linux and UNIX platforms, causes all processes running on all database partition servers to write call traceback to the syslog.

On Linux and UNIX platforms, these commands execute rah with certain implicit settings such as:

- Run in parallel at all computers
- Buffer command output in `/tmp/$USER/db2_kill`, `/tmp/$USER/db2_call_stack` respectively.

The command db2_call_stack is *not* available on Windows. Use the db2pd -stack command instead.

Specifying the rah and db2_all commands

You can specify rah command from the command line as the parameter, or in response to the prompt if you don't specify any parameter.

You should use the prompt method if the command contains the following special characters:

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

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

Note: On Linux and UNIX platforms, the command will be added to your command history just as if you typed it at the prompt.

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

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

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

Note:

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

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

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

if there is no input to be supplied.

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

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

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

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

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

Running commands in parallel (Linux, UNIX)

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

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

This process retrieves the output in two pieces:

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

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

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

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

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

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

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

```
export RAHCHECKBUF=no
```

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

Monitoring rah processes (Linux, UNIX)

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

About this task

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

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

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

```
kill <pid>
```

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

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

```
"  unsubstituted $ '
```

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

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

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

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

In the case of a multiple-logical-node-per-physical-node system, db2_all will favor sending the recursive invocation to distinct physical nodes, which will then rsh to other logical nodes on the same physical node, thus also reducing inter-physical-node traffic. (This point applies only to db2_all, not rah, since rah always sends only to distinct physical nodes.)

rah command prefix sequences

A prefix sequence is one or more special characters.

Type one or more prefix sequences immediately preceding the characters of the command without any intervening blanks. If you want to specify more than one sequence, you can type them in any order, but characters within any multicharacter sequence must be typed in order. If you type any prefix sequences, you must enclose the entire command, including the prefix sequences in double quotation marks, as in the following examples:

- On Linux and UNIX platforms:

```
rah "};ps -F pid,ppid,etime,args -u $USER"
```

- On Windows:

```
rah "||db2 get db cfg for sample"
```

The prefix sequences are:

Sequence	Purpose
`|`	Runs the commands in sequence in the background.
`|&`	Runs the commands in sequence in the background and terminates the command after all remote commands have completed, even if some processes are still running. This might be later if, for example, child processes (on Linux and UNIX platforms) or background processes (on Windows) are still running. In this case, the command starts a separate background process to retrieve any remote output generated after command termination and writes it back to the originating computer. **Note:** On Linux and UNIX platforms, specifying & degrades performance, because more rsh commands are required.
`||`	Runs the commands in parallel in the background.
`||&`	Runs the commands in parallel in the background and terminates the command after all remote commands have completed as described for the |& case above. **Note:** On Linux and UNIX platforms, specifying & degrades performance, because more rsh commands are required.
`;`	Same as ||& above. This is an alternative shorter form. **Note:** On Linux and UNIX platforms, specifying ; degrades performance relative to ||, because more rsh commands are required.
`]`	Prepends dot-execution of user's profile before executing command.

Note: Available on Linux and UNIX platforms only.

} Prepends dot-execution of file named in $RAHENV (probably `.kshrc`) before executing command.

Note: Available on Linux and UNIX platforms only.

]} Prepends dot-execution of user's profile followed by execution of file named in $RAHENV (probably `.kshrc`) before executing command.

Note: Available on Linux and UNIX platforms only.

) Suppresses execution of user's profile and of file named in $RAHENV.

Note: Available on Linux and UNIX platforms only.

' Echoes the command invocation to the computer.

< Sends to all the computers except this one.

<<-nnn<
Sends to all-but-database partition server *nnn* (all database partition servers in `db2nodes.cfg` except for node number *nnn*, see the first paragraph following the last prefix sequence in this table).

<<+nnn<
Sends to only database partition server *nnn* (the database partition server in `db2nodes.cfg` whose database partition number is *nnn*, see the first paragraph following the last prefix sequence in this table).

(blank character)
Runs the remote command in the background with `stdin`, `stdout`, and `stderr` all closed. This option is valid only when running the command in the background, that is, only in a prefix sequence which also includes \ or ;. It allows the command to complete much sooner (as soon as the remote command has been initiated). If you specify this prefix sequence on the rah command line, then either enclose the command in single quotation marks, or enclose the command in double quotation marks, and precede the prefix character by \ . For example,

```
rah '; mydaemon'
```

or

```
rah ";\ mydaemon"
```

When run as a background process, the rah command will never wait for any output to be returned.

> Substitutes occurrences of > with the computer name.

" Substitutes occurrences of () by the computer index, and substitutes occurrences of ## by the database partition number.

Note:

1. The computer index is a number that associated with a computer in the database system. If you are not running multiple logical partitions, the computer index for a computer corresponds to the database partition number for that computer in the node configuration file. To obtain the computer index for a computer in a multiple logical partition database environment, do not count duplicate entries for those computers that run multiple logical partitions. For example, if MACH1 is running two

logical partitions and MACH2 is also running two logical partitions, the database partition number for MACH3 is 5 in the node configuration file. The computer index for MACH3, however, would be 3.

On Windows, do not edit the node configuration file. To obtain the computer index, use the db2nlist command.

2. When " is specified, duplicates are not eliminated from the list of computers.

When using the <<-nnn< and <<+nnn< prefix sequences, *nnn* is any 1-, 2- or 3-digit database partition number which must match the *nodenum* value in the db2nodes.cfg file.

Note: Prefix sequences are considered to be part of the command. If you specify a prefix sequence as part of a command, you must enclose the entire command, including the prefix sequences, in double quotation marks.

Specifying the list of machines in a partitioned database environment

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

About this task

You can override this by:

- Specifying a pathname to the file that contains the list of computers by exporting (on Linux and UNIX platforms) or setting (on Windows) the environment variable RAHOSTFILE.
- Specifying the list explicitly, as a string of names separated by spaces, by exporting (on Linux and UNIX platforms) or setting (on Windows) the environment variable RAHOSTLIST.

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

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

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

If you are running DB2 Enterprise Server Edition with multiple logical database partition servers on one computer, your db2nodes.cfg file will contain multiple entries for that computer.

About this task

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

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

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

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

When specifying logical database partitions, you can restrict the list to include all logical database partitions except one, or only specify one using the <<-nnn< and <<+nnn< prefix sequences. You might want to do this if you want to run a command to catalog the database partition first, and when that has completed, run the same command at all other database partition servers, possibly in parallel. This is usually required when running the db2 restart database command. You will need to know the database partition number of the catalog partition to do this.

If you execute db2 restart database using the rah command, duplicate entries are eliminated from the list of computers. However if you specify the " prefix, then duplicates are not eliminated, because it is assumed that use of the " prefix implies sending to each database partition server, rather than to each computer.

Controlling the rah command

This topic lists the environment variables to control the rah command.

Table 137. Environment variables that control the rah command

Name	Meaning	Default
$RAHBUFDIR **Note:** Available on Linux and UNIX platforms only.	Directory for buffer	/tmp/$USER
$RAHBUFNAME **Note:** Available on Linux and UNIX platforms only.	Filename for buffer	rahout
$RAHOSTFILE (on Linux and UNIX platforms); RAHOSTFILE (on Windows)	File containing list of hosts	`db2nodes.cfg`
$RAHOSTLIST (on Linux and UNIX platforms); RAHOSTLIST (on Windows)	List of hosts as a string	extracted from $RAHOSTFILE
$RAHCHECKBUF **Note:** Available on Linux and UNIX platforms only.	If set to "no", bypass checks	not set
$RAHSLEEPTIME (on Linux and UNIX platforms); RAHSLEEPTIME (on Windows)	Time in seconds this script will wait for initial output from commands run in parallel	86400 seconds for db2_kill, 200 seconds for all others

Table 137. Environment variables that control the rah command (continued)

Name	Meaning	Default
$RAHWAITTIME (on Linux and UNIX platforms); RAHWAITTIME (on Windows)	On Windows, interval in seconds between successive checks that remote jobs are still running. On Linux and UNIX platforms, interval in seconds between successive checks that remote jobs are still running and `rah: waiting for pid> ...` messages. On all platforms, specify any positive integer. Prefix value with a leading zero to suppress messages, for example, export RAHWAITTIME=045. It is not necessary to specify a low value as rah does not rely on these checks to detect job completion.	45 seconds
$RAHENV **Note:** Available on Linux and UNIX platforms only.	Specifies filename to be executed if $RAHDOTFILES=E or K or PE or B	$ENV
$RAHUSER (on Linux and UNIX platforms); RAHUSER (on Windows)	On Linux and UNIX platforms, user ID under which the remote command is to be run. On Windows, the logon account associated with the DB2 Remote Command Service	$USER

Note: On Linux and UNIX platforms, the value of $RAHENV where rah is run is used, not the value (if any) set by the remote shell.

Specifying which . files run with rah (Linux and UNIX)

This topics lists the . files that are run if no prefix sequence is specified.

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

P .profile

E File named in $RAHENV (probably `.kshrc`)

K Same as E

PE .profile followed by file named in $RAHENV (probably `.kshrc`)

B Same as PE

N None (or Neither)

Note: If your login shell is not a Korn shell, any dot files you specify to be executed will be executed in a Korn shell process, and so must conform to Korn shell syntax. So, for example, if your login shell is a C shell, to have your `.cshrc` environment set up for commands executed by rah, you should either create a Korn shell `INSTHOME/.profile` equivalent to your `.cshrc` and specify in your `INSTHOME/.cshrc`:

```
setenv RAHDOTFILES P
```

or you should create a Korn shell INSTHOME/.kshrc equivalent to your .cshrc and specify in your INSTHOME/.cshrc:

```
setenv RAHDOTFILES E
setenv RAHENV INSTHOME/.kshrc
```

Also, it is essential that your .cshrc does not write to stdout if there is no tty (as when invoked by rsh). You can ensure this by enclosing any lines which write to stdout by, for example,

```
if { tty -s } then echo "executed .cshrc";
endif
```

Setting the default environment profile for rah on Windows

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

About this task

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

The file should have the following format:

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

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

Determining problems with rah (Linux, UNIX)

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

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

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

   This problem might be caused because:

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

2. You have received messages such as:

   - Login incorrect
   - Permission denied

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

   Note: You might have a need to have greater security regarding the transmission of passwords in clear text between database partitions. This will

depend on the remote shell program you are using. rah uses the remote shell program specified by the DB2RSHCMD registry variable. You can select between the two remote shell programs: ssh (for additional security), or rsh (or remsh for HP-UX). If this registry variable is not set, rsh (or remsh for HP-UX) is used.

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

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

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

   ```
   rsh somewher -l $USER db2_kill
   ```

   rah never completes.

   This is normal. rah starts background monitoring processes, which continue to run after it has exited. Those processes will normally persist until all processes associated with the command you ran have themselves terminated. In the case of db2_kill, this means termination of all database managers. You can terminate the monitoring processes by finding the process whose command is rahwaitfor and kill process_id>. Do not specify a signal number. Instead, use the default (15).

5. The output from rah is not displayed correctly, or rah incorrectly reports that $RAHBUFNAME does not exist, when multiple commands of rah were issued under the same $RAHUSER.

   This is because multiple concurrent executions of rah are trying to use the same buffer file (for example, $RAHBUFDIR/$RAHBUFNAME) for buffering the outputs. To prevent this problem, use a different $RAHBUFNAME for each concurrent rah command, for example in the following ksh:

   ```
   export RAHBUFNAME=rahout
   rah ";$command_1" &
   export RAHBUFNAME=rah2out
   rah ";$command_2" &
   ```

   or use a method that makes the shell choose a unique name automatically such as:

   ```
   RAHBUFNAME=rahout.$$ db2_all ";....."
   ```

   Whatever method you use, you must ensure you clean up the buffer files at some point if disk space is limited. rah does not erase a buffer file at the end of execution, although it will erase and then re-use an existing file the next time you specify the same buffer file.

6. You entered

   ```
   rah '"print from ()'
   ```

   and received the message:

   ```
   ksh: syntax error at line 1 : (' unexpected
   ```

   Prerequisites for the substitution of () and ## are:

   - Use db2_all, not rah.

- Ensure a RAHOSTFILE is used either by exporting RAHOSTFILE or by defaulting to your `/sqllib/db2nodes.cfg` file. Without these prerequisites, rah will leave the () and ## as is. You receive an error because the command print from () is not valid.

For a performance tip when running commands in parallel, use `|` rather than `|&`, and use `||` rather than `||&` or ; unless you truly need the function provided by &. Specifying & requires more remote shell commands and therefore degrades performance.

Load overview–partitioned database environments

In a multi-partition database, large amounts of data are located across many database partitions. Distribution keys are used to determine on which database partition each portion of the data resides. The data must be *distributed* before it can be loaded at the correct database partition.

When loading tables in a multi-partition database, the load utility can:

- Distribute input data in parallel
- Load data simultaneously on corresponding database partitions
- Transfer data from one system to another system

Loading data into a multi-partition database takes place in two phases: the *setup phase*, during which database partition resources such as table locks are acquired, and the *load phase*, during which the data is loaded into the database partitions. You can use the ISOLATE_PART_ERRS option of the LOAD command to select how errors are handled during either of these phases, and how errors on one or more of the database partitions affect the load operation on the database partitions that are not experiencing errors.

When loading data into a multi-partition database you can use one of the following modes:

PARTITION_AND_LOAD
Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

PARTITION_ONLY
Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. Each file includes a partition header that specifies how the data was distributed across the database partitions, and that the file can be loaded into the database using the LOAD_ONLY mode.

LOAD_ONLY
Data is assumed to be already distributed across the database partitions; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions.

LOAD_ONLY_VERIFY_PART
Data is assumed to be already distributed across the database partitions, but the data file does not contain a partition header. The distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dump file if the `dumpfile` file type modifier is specified. Otherwise, the rows are discarded.

If database partition violations exist on a particular loading database partition, a single warning is written to the load message file for that database partition.

ANALYZE
An optimal distribution map with even distribution across all database partitions is generated.

Concepts and terminology

The following terminology is used when discussing the behavior and operation of the load utility in a partitioned database environment with multiple database partitions:

- The *coordinator partition* is the database partition to which the user connects in order to perform the load operation. In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, it is assumed that the data file resides on this database partition unless the CLIENT option of the LOAD command is specified. Specifying CLIENT indicates that the data to be loaded resides on a remotely connected client.
- In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, the *pre-partitioning agent* reads the user data and distributes it in round-robin fashion to the *partitioning agents* which then distribute the data. This process is always performed on the coordinator partition. A maximum of one partitioning agent is allowed per database partition for any load operation.
- In the PARTITION_AND_LOAD, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, *load agents* run on each output database partition and coordinate the loading of data to that database partition.
- *Load to file agents* run on each output database partition during a PARTITION_ONLY load operation. They receive data from partitioning agents and write it to a file on their database partition.
- The SOURCEUSEREXIT option provides a facility through which the load utility can execute a customized script or executable, referred to herein as the *user exit*.

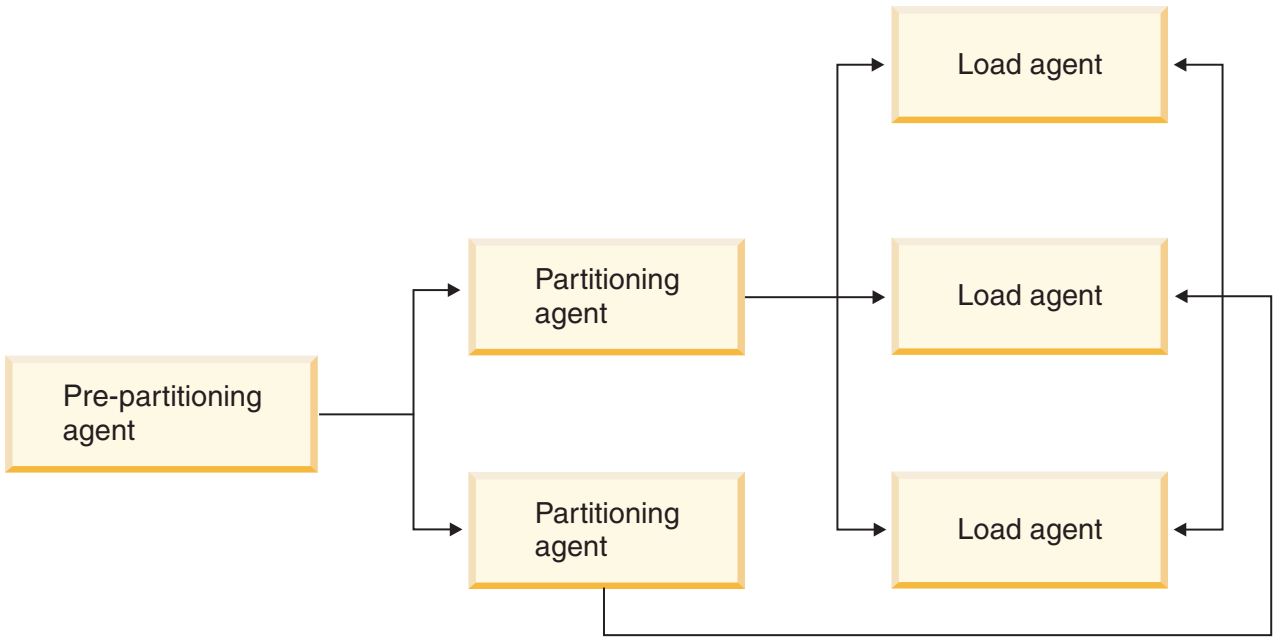


Figure 28. Partitioned Database Load Overview. The source data is read by the pre-partitioning agent, and approximately half of the data is sent to each of two partitioning agents which distribute the data and send it to one of three database partitions. The load agent at each database partition loads the data.

DB2 UDB Commands for Administrators

ADD DBPARTITIONNUM

Adds a database partition to a database partition server.

Scope

This command only affects the database partition server on which it is executed.

Authorization

One of the following:

- *sysadm*
- *sysctrl*

Required connection

None

Command syntax

```
►►—ADD DBPARTITIONNUM—┬—————————————————————————————————┬—►◄
                      ├—LIKE DBPARTITIONNUM—db-partition-number—┤
                      └—WITHOUT TABLESPACES———————————————————┘
```

Command parameters

LIKE DBPARTITIONNUM *db-partition-number*

Specifies that the containers for the new system temporary table spaces are the same as the containers of the database at the database partition server specified by *db-partition-number*. The database partition server specified must already be defined in the `db2nodes.cfg` file.

For system temporary table spaces that are defined to use automatic storage (in other words, system temporary table spaces that were created with the `MANAGED BY AUTOMATIC STORAGE` clause of the `CREATE TABLESPACE` statement or where no `MANAGED BY CLAUSE` was specified at all), the containers will not necessarily match those from the partition specified. Instead, containers will automatically be assigned by the database manager based on the storage paths that are associated with the database. This may or may not result in the same containers being used on these two partitions.

WITHOUT TABLESPACES

Specifies that containers for the system temporary table spaces are not created for any of the database partitions. The ALTER TABLESPACE statement must be used to add system temporary table space containers to each database partition before the database can be used.

If no option is specified, containers for the system temporary table spaces will be the same as the containers on the catalog partition for each database. The catalog partition can be a different database partition for each database in the partitioned database environment. This option is ignored for system temporary table spaces that are defined to use automatic storage (in other words, system temporary table spaces that were created with the `MANAGED BY AUTOMATIC STORAGE` clause of the `CREATE TABLESPACE` statement or where no `MANAGED BY CLAUSE` was specified at all). For these table spaces, there is no way to defer container creation. Containers will automatically be assigned by the database manager based on the storage paths that are associated with the database.

Usage notes

This command should only be used if a database partition server is added to an environment that has one database and that database is not cataloged at the time of the add partition operation. In this situation, because the database is not cataloged, the add partition operation does not recognize the database, and does not create a database partition for the database on the new database partition server. Any attempt to connect to the database partition on the new database partition server results in an error. The database must first be cataloged before the ADD DBPARTITIONNUM command can be used to create the database partition for the database on the new database partition server.

This command should not be used if the environment has more than one database and at least one of the databases is cataloged at the time of the add partition operation. In this situation, use the AT DBPARTITIONNUM parameter of the CREATE DATABASE command to create a database partition for each database that was not cataloged at the time of the add partition operation. Each uncataloged database must first be cataloged before the CREATE DATABASE command can be used to create the database partition for the database on the new database partition server.

Before adding a new database partition, ensure that there is sufficient storage for the containers that must be created.

The add database partition server operation creates an empty database partition for every database that exists in the instance. The configuration parameters for the new database partitions are set to the default values.

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

If an add database partition server operation fails while creating a database partition locally, it enters a clean-up phase, in which it locally drops all databases that have been created. This means that the database partitions are removed only from the database partition server being added. Existing database partitions remain unaffected on all other database partition servers. If the clean-up phase fails, no further clean up is done, and an error is returned.

The database partitions on the new database partition cannot contain user data until after the ALTER DATABASE PARTITION GROUP statement has been used to add the database partition to a database partition group.

This command will fail if a create database or a drop database operation is in progress. The command can be reissued once the competing operation has completed.

To determine whether or not a database is enabled for automatic storage, ADD DBPARTITIONNUM has to communicate with the catalog partition for each of the databases in the instance. If automatic storage is enabled then 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, ADD DBPARTITIONNUM might have to communicate with another database partition server to retrieve the table space definitions for the database partitions that reside on that server. The **start_stop_time** database manager configuration parameter is used to specify the time, in minutes, by which the other database partition server must respond with the automatic storage and table space definitions. If this time is exceeded, the command fails. If this situation occurs, increase the value of **start_stop_time**, and reissue the command.

Compatibilities

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.

BACKUP DATABASE

Creates a backup copy of a database or a table space.

For information on the backup operations supported by DB2 database systems between different operating systems and hardware platforms, see “Backup and restore operations between different operating systems and hardware platforms”.

Scope

In a partitioned database environment, if no database partitions are specified, this command affects only the database partition on which it is executed.

If the option to perform a partitioned backup is specified, the command can be called only on the catalog node. If the option specifies that all database partition servers are to be backed up, it affects all database partition servers that are listed in the `db2nodes.cfg` file. Otherwise, it affects the database partition servers that are specified on the command.

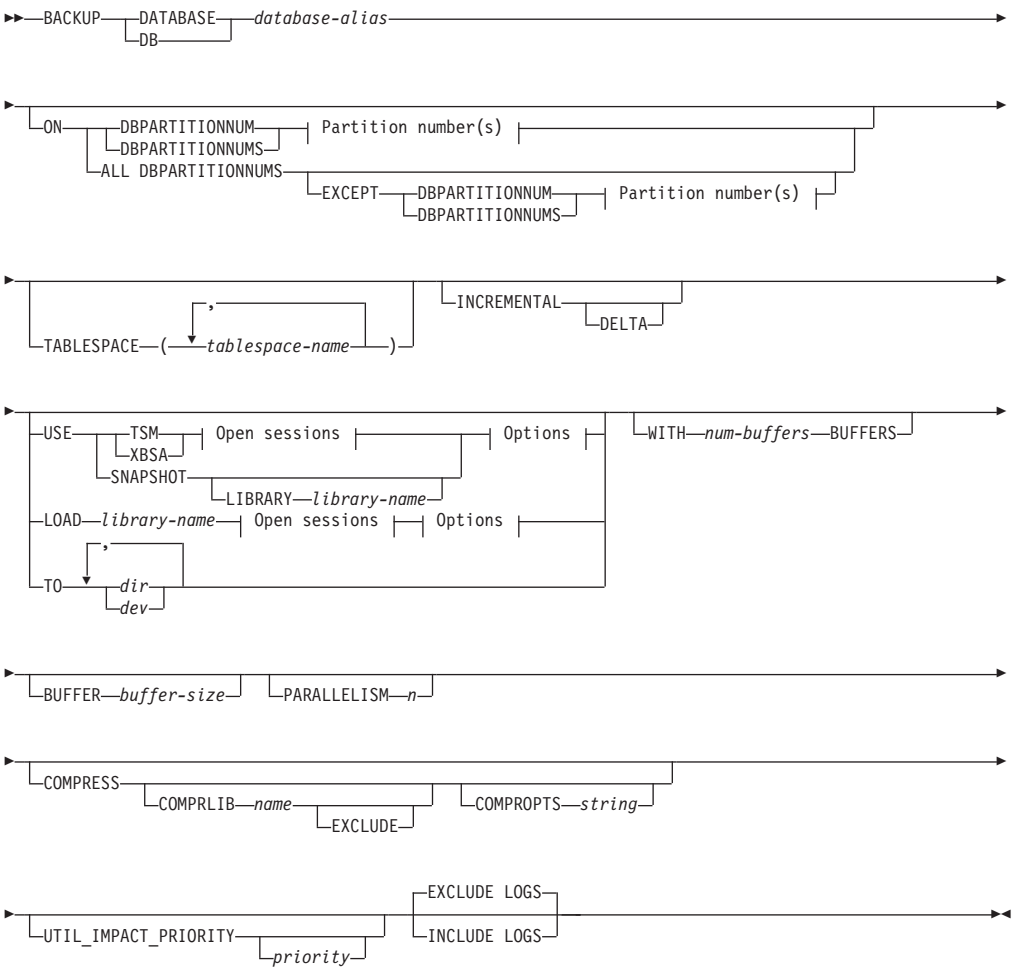
Authorization

One of the following:

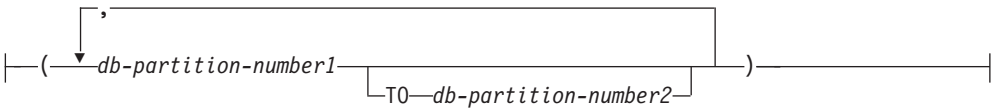
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection

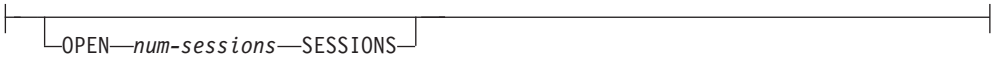
Command syntax



Partition number(s):



Open sessions:



Options:

Command parameters

DATABASE | DB *database-alias*
Specifies the alias of the database to back up.

ON Backup the database on a set of database partitions.

DBPARTITIONNUM *db-partition-number1*
Specifies a database partition number in the database partition list.

DBPARTITIONNUMS *db-partition-number1* **TO** *db-partition-number2*
Specifies a range of database partition numbers, so that all partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

ALL DBPARTITIONNUMS
Specifies that the database is to be backed up on all partitions specified in the `db2nodes.cfg` file.

EXCEPT
Specifies that the database is to be backed up on all partitions specified in the `db2nodes.cfg` file, except those specified in the database partition list.

DBPARTITIONNUM *db-partition-number1*
Specifies a database partition number in the database partition list.

DBPARTITIONNUMS *db-partition-number1* **TO** *db-partition-number2*
Specifies a range of database partition numbers, so that all partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

TABLESPACE *tablespace-name*
A list of names used to specify the table spaces to be backed up.

ONLINE

INCREMENTAL

Specifies a cumulative (incremental) backup image. An incremental backup image is a copy of all database data that has changed since the most recent successful, full backup operation.

DELTA

Specifies a non-cumulative (delta) backup image. A delta backup image is a copy of all database data that has changed since the most recent successful backup operation of any type.

USE

TSM Specifies that the backup is to use Tivoli® Storage Manager (TSM) output.

XBSA Specifies that the XBSA interface is to be used. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.

SNAPSHOT

Specifies that a snapshot backup is to be taken.

You cannot use the SNAPSHOT parameter with any of the following parameters:

- TABLESPACE
- INCREMENTAL
- WITH *num-buffers* BUFFERS
- BUFFER
- PARALLELISM
- COMPRESS
- UTIL_IMPACT_PRIORITY
- SESSIONS

The default behavior for a snapshot backup is a FULL DATABASE OFFLINE backup of all paths that make up the database including all containers, local volume directory, database path (DBPATH), and primary log and mirror log paths (INCLUDE LOGS is the default for all snapshot backups unless EXCLUDE LOGS is explicitly stated).

LIBRARY *library-name*

Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage SAN Volume Controller
- IBM Enterprise Storage Server Model 800
- IBM System Storage DS6000
- IBM System Storage DS8000
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS

If you have other storage hardware, and a DB2 ACS API driver for that storage hardware, you can use the LIBRARY parameter to specify the DB2 ACS API driver.

The value of the LIBRARY parameter is a fully-qualified library file name.

OPTIONS

"options-string"
Specifies options to be used for the backup operation. The string will be passed to the DB2 ACS API driver exactly as it was entered, without the double quotation marks. You cannot use the **VENDOROPT** database configuration parameter to specify vendor-specific options for snapshot backup operations. You must use the OPTIONS parameter of the backup utilities instead.

@ file-name
Specifies that the options to be used for the backup operation are contained in a file located on the DB2 server. The string will be passed to the vendor support library. The file must be a fully qualified file name.

OPEN *num-sessions* **SESSIONS**
The number of I/O sessions to be created between DB2 and TSM or another backup vendor product. This parameter has no effect when backing up to tape, disk, or other local device.

TO *dir* | *dev*
A list of directory or tape device names. The full path on which the directory resides must be specified. This target directory or device must exist on the database server.

In a partitioned database, the target directory or device must exist on all database partitions, and can optionally be a shared path. The directory or device name may be specified using a database partition expression. For more information about database partition expressions, see *Automatic storage databases*.

This parameter can be repeated to specify the target directories and devices that the backup image will span. If more than one target is specified (target1, target2, and target3, for example), target1 will be opened first. The media header and special files (including the configuration file, table space table, and history file) are placed in target1. All remaining targets are opened, and are then used in parallel during the backup operation. Because there is no general tape support on Windows operating systems, each type of tape device requires a unique device driver.

If the tape system does not support the ability to uniquely reference a backup image, it is recommended that multiple backup copies of the same database not be kept on the same tape.

LOAD *library-name*
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. It can contain the full path. If the full path is not given, it will default to the path on which the user exit program resides.

WITH *num-buffers* **BUFFERS**
The number of buffers to be used. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value. However, when creating a backup to multiple locations, a larger number of buffers can be used to improve performance.

BUFFER *buffer-size*

The size, in 4 KB pages, of the buffer used when building the backup image. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value. The minimum value for this parameter is 8 pages.

If using tape with variable block size, reduce the buffer size to within the range that the tape device supports. Otherwise, the backup operation might succeed, but the resulting image might not be recoverable.

With most versions of Linux, using DB2's default buffer size for backup operations to a SCSI tape device results in error SQL2025N, reason code 75. To prevent the overflow of Linux internal SCSI buffers, use this formula:

```
bufferpages <= ST_MAX_BUFFERS * ST_BUFFER_BLOCKS / 4
```

where *bufferpages* is the value you want to use with the BUFFER parameter, and `ST_MAX_BUFFERS` and `ST_BUFFER_BLOCKS` are defined in the Linux kernel under the `drivers/scsi` directory.

PARALLELISM *n*

Determines the number of table spaces which can be read in parallel by the backup utility. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value.

UTIL_IMPACT_PRIORITY *priority*

Specifies that the backup will run in throttled mode, with the priority specified. Throttling allows you to regulate the performance impact of the backup operation. Priority can be any number between 1 and 100, with 1 representing the lowest priority, and 100 representing the highest priority. If the UTIL_IMPACT_PRIORITY keyword is specified with no priority, the backup will run with the default priority of 50. If UTIL_IMPACT_PRIORITY is not specified, the backup will run in unthrottled mode. An impact policy must be defined by setting the *util_impact_lim* configuration parameter for a backup to run in throttled mode.

COMPRESS

Indicates that the backup is to be compressed.

COMPRLIB *name*

Indicates the name of the library to be used to perform the compression (e.g., `db2compr.dll` for Windows; `libdb2compr.so` for Linux/UNIX systems). The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, the default DB2 compression library will be used. If the specified library cannot be loaded, the backup will fail.

EXCLUDE

Indicates that the compression library will not be stored in the backup image.

COMPROPTS *string*

Describes a block of binary data that will be passed to the initialization routine in the compression library. DB2 will pass this string directly from the client to the server, so any issues of byte reversal or code page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents

of string with the contents of this file and will pass this new value to the initialization routine instead. The maximum length for *string* is 1024 bytes.

EXCLUDE LOGS
Specifies that the backup image should not include any log files. When performing an offline backup operation, logs are excluded whether or not this option is specified, with the exception of snapshot backups.

INCLUDE LOGS
Specifies that the backup image should include the range of log files required to restore and roll forward this image to some consistent point in time. This option is not valid for an offline backup, with the exception of snapshot backups where this option is the default unless explicitly told to exclude.

WITHOUT PROMPTING

Examples

Usage notes

The data in a backup cannot be protected by the database server. Make sure that backups are properly safeguarded, particularly if the backup contains LBAC-protected data.

When backing up to tape, use of a variable block size is currently not supported. If you must use this option, ensure that you have well tested procedures in place that enable you to recover successfully, using backup images that were created with a variable block size.

When using a variable block size, you must specify a backup buffer size that is less than or equal to the maximum limit for the tape devices that you are using. For optimal performance, the buffer size must be equal to the maximum block size limit of the device being used.

Snapshot backups should be complemented with regular disk backups in case of failure in the filer/storage system.

As you regularly backup your database, you might accumulate very large database backup images, many database logs and load copy images, all of which might be taking up a large amount of disk space. Refer to “Managing recovery objects” for information on how to manage these recovery objects.

BACKUP DATABASE

Creates a backup copy of a database or a table space.

For information on the backup operations supported by DB2 database systems between different operating systems and hardware platforms, see “Backup and restore operations between different operating systems and hardware platforms”.

Scope

In a partitioned database environment, if no database partitions are specified, this command affects only the database partition on which it is executed.

If the option to perform a partitioned backup is specified, the command can be called only on the catalog node. If the option specifies that all database partition servers are to be backed up, it affects all database partition servers that are listed in the db2nodes.cfg file. Otherwise, it affects the database partition servers that are specified on the command.

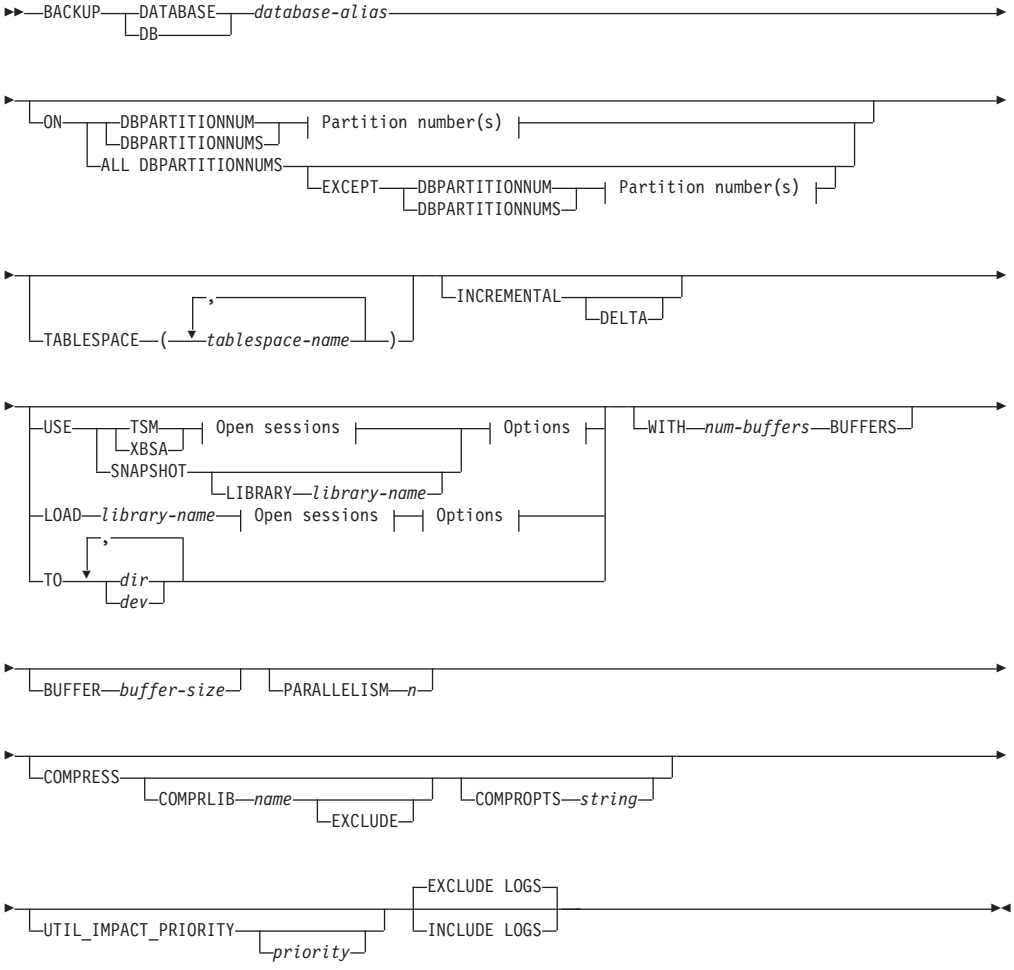
Authorization

One of the following:

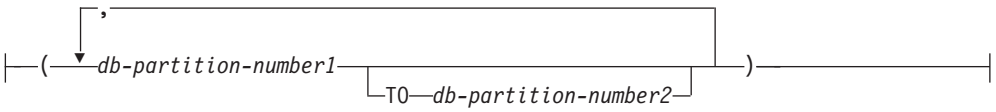
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection

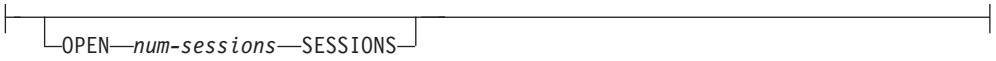
Command syntax



Partition number(s):



Open sessions:



Options:

Command parameters

DATABASE | DB *database-alias*
Specifies the alias of the database to back up.

ON Backup the database on a set of database partitions.

DBPARTITIONNUM *db-partition-number1*
Specifies a database partition number in the database partition list.

DBPARTITIONNUMS *db-partition-number1* **TO** *db-partition-number2*
Specifies a range of database partition numbers, so that all partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

ALL DBPARTITIONNUMS
Specifies that the database is to be backed up on all partitions specified in the `db2nodes.cfg` file.

EXCEPT
Specifies that the database is to be backed up on all partitions specified in the `db2nodes.cfg` file, except those specified in the database partition list.

DBPARTITIONNUM *db-partition-number1*
Specifies a database partition number in the database partition list.

DBPARTITIONNUMS *db-partition-number1* **TO** *db-partition-number2*
Specifies a range of database partition numbers, so that all partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

TABLESPACE *tablespace-name*
A list of names used to specify the table spaces to be backed up.

ONLINE

INCREMENTAL

Specifies a cumulative (incremental) backup image. An incremental backup image is a copy of all database data that has changed since the most recent successful, full backup operation.

DELTA

Specifies a non-cumulative (delta) backup image. A delta backup image is a copy of all database data that has changed since the most recent successful backup operation of any type.

USE

TSM Specifies that the backup is to use Tivoli Storage Manager (TSM) output.

XBSA Specifies that the XBSA interface is to be used. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.

SNAPSHOT

Specifies that a snapshot backup is to be taken.

You cannot use the SNAPSHOT parameter with any of the following parameters:

- TABLESPACE
- INCREMENTAL
- WITH *num-buffers* BUFFERS
- BUFFER
- PARALLELISM
- COMPRESS
- UTIL_IMPACT_PRIORITY
- SESSIONS

The default behavior for a snapshot backup is a FULL DATABASE OFFLINE backup of all paths that make up the database including all containers, local volume directory, database path (DBPATH), and primary log and mirror log paths (INCLUDE LOGS is the default for all snapshot backups unless EXCLUDE LOGS is explicitly stated).

LIBRARY *library-name*

Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage SAN Volume Controller
- IBM Enterprise Storage Server Model 800
- IBM System Storage DS6000
- IBM System Storage DS8000
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS

If you have other storage hardware, and a DB2 ACS API driver for that storage hardware, you can use the LIBRARY parameter to specify the DB2 ACS API driver.

The value of the LIBRARY parameter is a fully-qualified library file name.

OPTIONS

"options-string"
Specifies options to be used for the backup operation. The string will be passed to the DB2 ACS API driver exactly as it was entered, without the double quotation marks. You cannot use the **VENDOROPT** database configuration parameter to specify vendor-specific options for snapshot backup operations. You must use the OPTIONS parameter of the backup utilities instead.

@ file-name
Specifies that the options to be used for the backup operation are contained in a file located on the DB2 server. The string will be passed to the vendor support library. The file must be a fully qualified file name.

OPEN *num-sessions* **SESSIONS**
The number of I/O sessions to be created between DB2 and TSM or another backup vendor product. This parameter has no effect when backing up to tape, disk, or other local device.

TO *dir* | *dev*
A list of directory or tape device names. The full path on which the directory resides must be specified. This target directory or device must exist on the database server.

In a partitioned database, the target directory or device must exist on all database partitions, and can optionally be a shared path. The directory or device name may be specified using a database partition expression. For more information about database partition expressions, see *Automatic storage databases*.

This parameter can be repeated to specify the target directories and devices that the backup image will span. If more than one target is specified (target1, target2, and target3, for example), target1 will be opened first. The media header and special files (including the configuration file, table space table, and history file) are placed in target1. All remaining targets are opened, and are then used in parallel during the backup operation. Because there is no general tape support on Windows operating systems, each type of tape device requires a unique device driver.

If the tape system does not support the ability to uniquely reference a backup image, it is recommended that multiple backup copies of the same database not be kept on the same tape.

LOAD *library-name*
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. It can contain the full path. If the full path is not given, it will default to the path on which the user exit program resides.

WITH *num-buffers* **BUFFERS**
The number of buffers to be used. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value. However, when creating a backup to multiple locations, a larger number of buffers can be used to improve performance.

BUFFER *buffer-size*

The size, in 4 KB pages, of the buffer used when building the backup image. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value. The minimum value for this parameter is 8 pages.

If using tape with variable block size, reduce the buffer size to within the range that the tape device supports. Otherwise, the backup operation might succeed, but the resulting image might not be recoverable.

With most versions of Linux, using DB2's default buffer size for backup operations to a SCSI tape device results in error SQL2025N, reason code 75. To prevent the overflow of Linux internal SCSI buffers, use this formula:

```
bufferpages <= ST_MAX_BUFFERS * ST_BUFFER_BLOCKS / 4
```

where *bufferpages* is the value you want to use with the BUFFER parameter, and `ST_MAX_BUFFERS` and `ST_BUFFER_BLOCKS` are defined in the Linux kernel under the `drivers/scsi` directory.

PARALLELISM *n*

Determines the number of table spaces which can be read in parallel by the backup utility. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value.

UTIL_IMPACT_PRIORITY *priority*

Specifies that the backup will run in throttled mode, with the priority specified. Throttling allows you to regulate the performance impact of the backup operation. Priority can be any number between 1 and 100, with 1 representing the lowest priority, and 100 representing the highest priority. If the UTIL_IMPACT_PRIORITY keyword is specified with no priority, the backup will run with the default priority of 50. If UTIL_IMPACT_PRIORITY is not specified, the backup will run in unthrottled mode. An impact policy must be defined by setting the *util_impact_lim* configuration parameter for a backup to run in throttled mode.

COMPRESS

Indicates that the backup is to be compressed.

COMPRLIB *name*

Indicates the name of the library to be used to perform the compression (e.g., `db2compr.dll` for Windows; `libdb2compr.so` for Linux/UNIX systems). The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, the default DB2 compression library will be used. If the specified library cannot be loaded, the backup will fail.

EXCLUDE

Indicates that the compression library will not be stored in the backup image.

COMPROPTS *string*

Describes a block of binary data that will be passed to the initialization routine in the compression library. DB2 will pass this string directly from the client to the server, so any issues of byte reversal or code page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents

of string with the contents of this file and will pass this new value to the initialization routine instead. The maximum length for *string* is 1024 bytes.

EXCLUDE LOGS
Specifies that the backup image should not include any log files. When performing an offline backup operation, logs are excluded whether or not this option is specified, with the exception of snapshot backups.

INCLUDE LOGS
Specifies that the backup image should include the range of log files required to restore and roll forward this image to some consistent point in time. This option is not valid for an offline backup, with the exception of snapshot backups where this option is the default unless explicitly told to exclude.

WITHOUT PROMPTING

Examples

Usage notes

The data in a backup cannot be protected by the database server. Make sure that backups are properly safeguarded, particularly if the backup contains LBAC-protected data.

When backing up to tape, use of a variable block size is currently not supported. If you must use this option, ensure that you have well tested procedures in place that enable you to recover successfully, using backup images that were created with a variable block size.

When using a variable block size, you must specify a backup buffer size that is less than or equal to the maximum limit for the tape devices that you are using. For optimal performance, the buffer size must be equal to the maximum block size limit of the device being used.

Snapshot backups should be complemented with regular disk backups in case of failure in the filer/storage system.

As you regularly backup your database, you might accumulate very large database backup images, many database logs and load copy images, all of which might be taking up a large amount of disk space. Refer to “Managing recovery objects” for information on how to manage these recovery objects.

BIND

Invokes the bind utility, which prepares SQL statements stored in the bind file generated by the precompiler, and creates a package that is stored in the database.

Scope

This command can be issued from any database partition in `db2nodes.cfg`. It updates the database catalogs on the catalog database partition. Its effects are visible to all database partitions.

Authorization

One of the following authorizations:

- *dbadm* authority
- If EXPLAIN ONLY is specified, EXPLAIN authority or an authority that implicitly includes EXPLAIN is sufficient.
- If a package does not exist, BINDADD authority and:
  - If the schema name of the package does not exist, IMPLICIT_SCHEMA authority on the database.
  - If the schema name of the package does exist, CREATEIN privilege on the schema.
- If the package exists, one of the following privileges:
  - ALTERIN privilege on the schema
  - BIND privilege on the package

In addition, if capturing explain information using the EXPLAIN or the EXPLSNAP clause, one of the following authorizations is required:

- INSERT privilege on the explain tables
- DATAACCESS authority

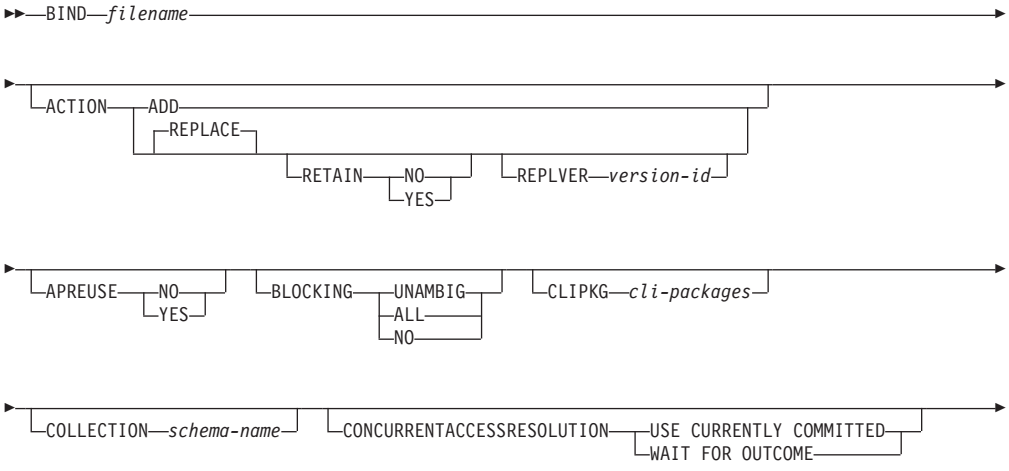
The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements.

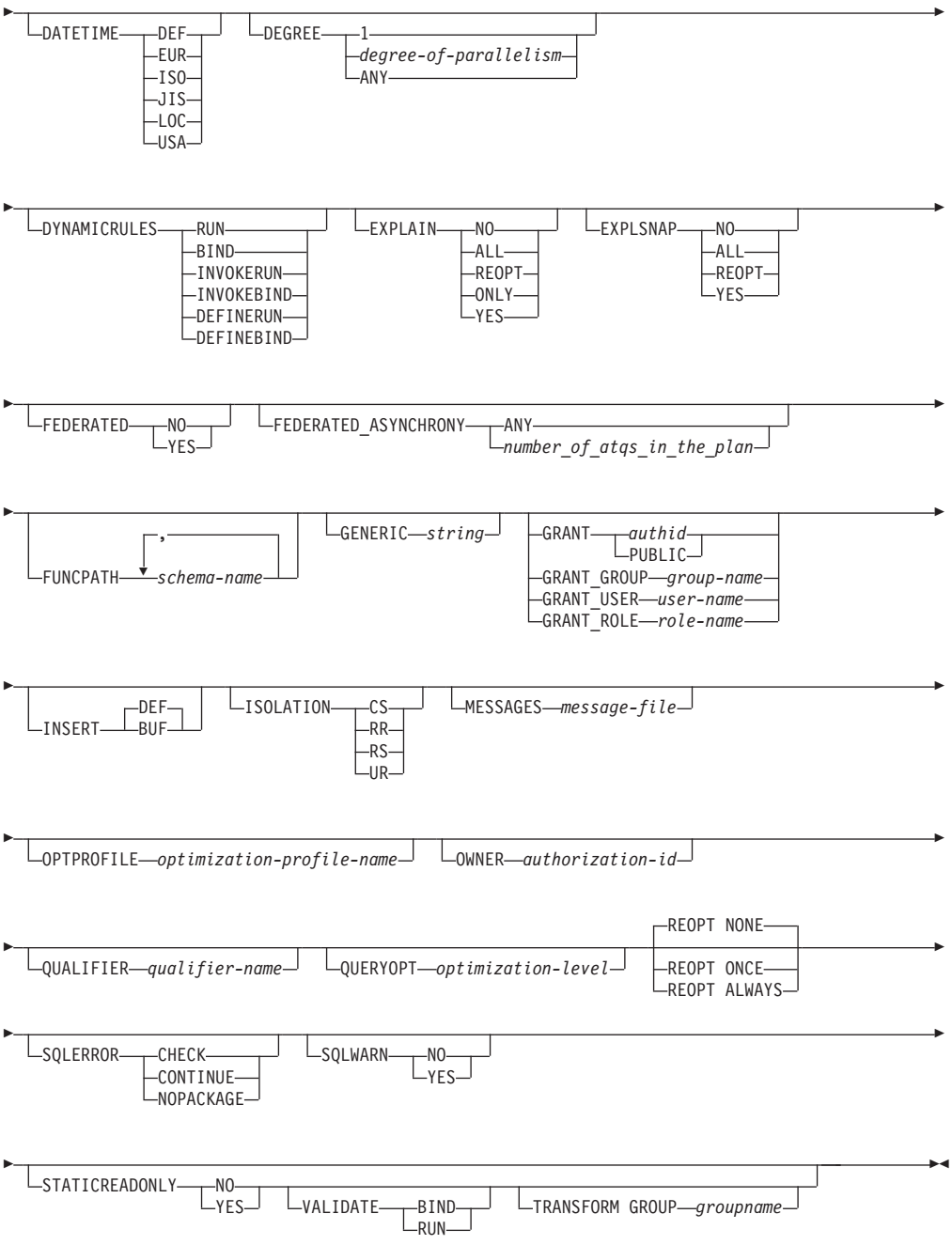
Required connection

Database. If implicit connect is enabled, a connection to the default database is established.

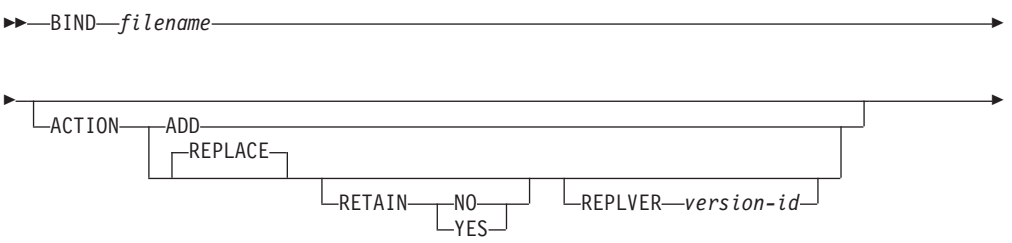
Command syntax

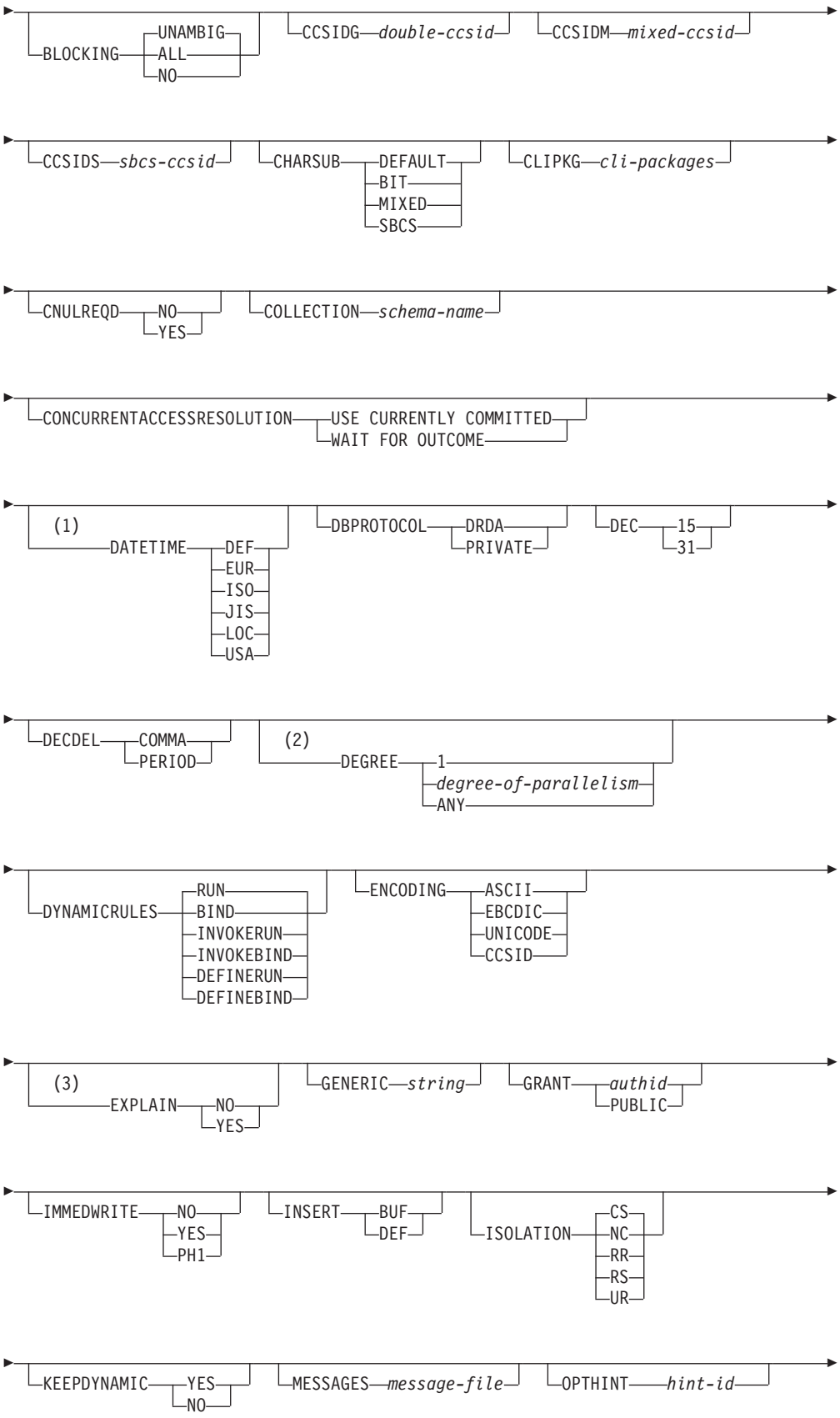
For DB2 for Linux, Windows and UNIX

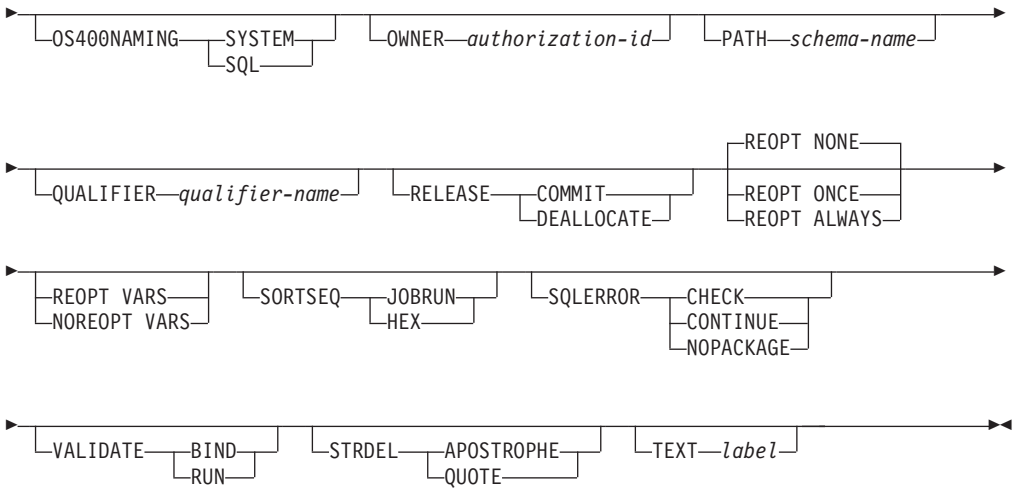




For DB2 on servers other than Linux, Windows and UNIX




BLOCKING
UNAMBIG
ALL
NO
CCSIDG
double-ccsid
CCSIDM
mixed-ccsid
CCSIDS
sbcs-ccsid
CHARSUB
DEFAULT
BIT
MIXED
SBCS
CLIPKG
cli-packages
CNULREQD
NO
YES
COLLECTION
schema-name
CONCURRENTACCESSRESOLUTION
USE CURRENTLY COMMITTED
WAIT FOR OUTCOME
(1)
DATETIME
DEF
EUR
ISO
JIS
LOC
USA
DBPROTOCOL
DRDA
PRIVATE
DEC
15
31
DECDEL
COMMA
PERIOD
(2)
DEGREE
1
degree-of-parallelism
ANY
DYNAMICRULES
RUN
BIND
INVOKERUN
INVOKEBIND
DEFINERUN
DEFINEBIND
ENCODING
ASCII
EBCDIC
UNICODE
CCSID
(3)
EXPLAIN
NO
YES
GENERIC
string
GRANT
authid
PUBLIC
IMMEDWRITE
NO
YES
PH1
INSERT
BUF
DEF
ISOLATION
CS
NC
RR
RS
UR
KEEPDYNAMIC
YES
NO
MESSAGES
message-file
OPTHINT
hint-id



Notes:

1 If the server does not support the DATETIME DEF option, it is mapped to DATETIME ISO.

2 The DEGREE option is only supported by DRDA Level 2 Application Servers.

3 DRDA defines the EXPLAIN option to have the value YES or NO. If the server does not support the EXPLAIN YES option, the value is mapped to EXPLAIN ALL.

Command parameters

filename

Specifies the name of the bind file that was generated when the application program was precompiled, or a list file containing the names of several bind files. Bind files have the extension `.bnd`. The full path name can be specified.

If a list file is specified, the @ character must be the first character of the list file name. The list file can contain several lines of bind file names. Bind files listed on the same line must be separated by plus (+) characters, but a + cannot appear in front of the first file listed on each line, or after the last bind file listed. For example,

```
/u/smith/sqllib/bnd/@all.lst
```

is a list file that contains the following bind files:

```
mybind1.bnd+mybind.bnd2+mybind3.bnd+
      mybind4.bnd+mybind5.bnd+
      mybind6.bnd+
      mybind7.bnd
```

ACTION

Indicates whether the package can be added or replaced.

ADD Indicates that the named package does not exist, and that a new package is to be created. If the package already exists, execution stops, and a diagnostic error message is returned.

REPLACE

Indicates that the existing package is to be replaced by a new one with the same package name and creator. This is the default value for the ACTION option.

RETAIN

Indicates whether BIND and EXECUTE authorities are to be preserved when a package is replaced. If ownership of the package changes, the new owner grants the BIND and EXECUTE authority to the previous package owner.

NO Does not preserve BIND and EXECUTE authorities when a package is replaced. This value is not supported by DB2.

YES Preserves BIND and EXECUTE authorities when a package is replaced. This is the default value.

REPLVER *version-id*

Replaces a specific version of a package. The version identifier specifies which version of the package is to be replaced. If the specified version does not exist, an error is returned. If the REPLVER option of REPLACE is not specified, and a package already exists that matches the package name, creator, and version of the package being bound, that package will be replaced; if not, a new package will be added.

APREUSE

Specifies whether static SQL access plans are to be reused. When this option is enabled, the query compiler will attempt to reuse the access plans for the statement in any existing packages during the bind and during future implicit and explicit rebinds.

YES The query compiler will attempt to reuse the access plans for the statements in the package. If there is an existing package, the query compiler will attempt to reuse the access plan for every statement that can be matched with a statement in the new bind file. For a statement to match, the statement text must be identical and the section number for the statement in the existing package must match what the section number will be for the statement in the new package.

NO The query compiler will not attempt to reuse access plans for the statements in the package. This is the default setting.

BLOCKING

Specifies the type of row blocking for cursors. The blocking of row data that contains references to LOB column data types is also supported in environments where the Database Partitioning Feature (DPF) is enabled.

ALL For cursors that are specified with the FOR READ ONLY clause or cursors not specified as FOR UPDATE, blocking occurs.

Ambiguous cursors are treated as read-only.

NO Blocking does not occur for any cursor.

For the definition of a read-only cursor and an ambiguous cursor, refer to *DECLARE CURSOR statement*.

Ambiguous cursors are treated as updatable.

UNAMBIG

For cursors that are specified with the FOR READ ONLY clause, blocking occurs.

Cursors that are not declared with the FOR READ ONLY or FOR UPDATE clause which are not ambiguous and are read-only will be blocked. Ambiguous cursors will not be blocked.

Ambiguous cursors are treated as updatable.

CCSIDG *double-ccsid*
An integer specifying the coded character set identifier (CCSID) to be used for double byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CCSIDM *mixed-ccsid*
An integer specifying the coded character set identifier (CCSID) to be used for mixed byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CCSIDS *sbcs-ccsid*
An integer specifying the coded character set identifier (CCSID) to be used for single byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CHARSUB
Designates the default character sub-type that is to be used for column definitions in CREATE and ALTER TABLE SQL statements. This DRDA precompile/bind option is not supported by DB2 Database for Linux, UNIX, and Windows.

BIT Use the FOR BIT DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

DEFAULT
Use the target system defined default in all new character columns for which an explicit sub-type is not specified.

MIXED
Use the FOR MIXED DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

SBCS Use the FOR SBCS DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

CLIPKG *cli-packages*
An integer between 3 and 30 specifying the number of CLI large packages to be created when binding CLI bind files against a database.

CNULREQD
This option is related to the LANGLEVEL precompile option, which is not supported by DRDA. It is valid only if the bind file is created from a C or a C++ application. This DRDA bind option is not supported by DB2 Database for Linux, UNIX, and Windows.

NO The application was coded on the basis of the LANGLEVEL SAA1 precompile option with respect to the null terminator in C string host variables.

YES The application was coded on the basis of the LANGLEVEL MIA precompile option with respect to the null terminator in C string host variables.

COLLECTION *schema-name*
Specifies a 128-byte collection identifier for the package. If not specified, the authorization identifier for the user processing the package is used.

CONCURRENTACCESSRESOLUTION
Specifies the concurrent access resolution to use for statements in the package.

USE CURRENTLY COMMITTED
Specifies that the database manager can use the currently committed version of the data for applicable scans when it is in the process of being updated or deleted. Rows in the process of being inserted can be skipped. This clause applies when the isolation level in effect is Cursor Stability or Read Stability (for Read Stability it skips uncommited inserts only) and is ignored otherwise. Applicable scans include read-only scans that can be part of a read-only statement as well as a non read-only statement. The settings for the registry variables **DB2_EVALUNCOMMITTED**, **DB2_SKIPDELETED**, and **DB2_SKIPINSERTED** no longer apply.

WAIT FOR OUTCOME
Specifies Cursor Stability and higher scans to wait for the commit or rollback when encountering data in the process of being updated. Rows in the process of being inserted or deleted rows are not skipped. The settings for the registry variables **DB2_EVALUNCOMMITTED**, **DB2_SKIPDELETED**, and **DB2_SKIPINSERTED** no longer apply.

DATETIME
Specifies the date and time format to be used.

DEF Use a date and time format associated with the territory code of the database.

EUR Use the IBM standard for Europe date and time format.

ISO Use the date and time format of the International Standards Organization.

JIS Use the date and time format of the Japanese Industrial Standard.

LOC Use the date and time format in local form associated with the territory code of the database.

USA Use the IBM standard for U.S. date and time format.

DBPROTOCOL
Specifies what protocol to use when connecting to a remote site that is identified by a three-part name statement. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

DEC Specifies the maximum precision to be used in decimal arithmetic operations. This DRDA precompile/bind option is not supported by DB2

Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

15 15-digit precision is used in decimal arithmetic operations.

31 31-digit precision is used in decimal arithmetic operations.

DECDEL
Designates whether a period (.) or a comma (,) will be used as the decimal point indicator in decimal and floating point literals. This DRDA precompile/bind option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

COMMA
Use a comma (,) as the decimal point indicator.

PERIOD
Use a period (.) as the decimal point indicator.

DEGREE
Specifies the degree of parallelism for the execution of static SQL statements in an SMP system. This option does not affect CREATE INDEX parallelism.

1 The execution of the statement will not use parallelism.

degree-of-parallelism
Specifies the degree of parallelism with which the statement can be executed, a value between 2 and 32 767 (inclusive).

ANY Specifies that the execution of the statement can involve parallelism using a degree determined by the database manager.

DYNAMICRULES
Defines which rules apply to dynamic SQL at run time for the initial setting of the values used for authorization ID and for the implicit qualification of unqualified object references.

RUN Specifies that the authorization ID of the user executing the package is to be used for authorization checking of dynamic SQL statements. The authorization ID will also be used as the default package qualifier for implicit qualification of unqualified object references within dynamic SQL statements. This is the default value.

BIND Specifies that all of the rules that apply to static SQL for authorization and qualification are to be used at run time. That is, the authorization ID of the package owner is to be used for authorization checking of dynamic SQL statements, and the default package qualifier is to be used for implicit qualification of unqualified object references within dynamic SQL statements.

DEFINERUN
If the package is used within a routine context, the authorization ID of the routine definer is to be used for authorization checking and for implicit qualification of unqualified object references within dynamic SQL statements within the routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES RUN.

DEFINEBIND
If the package is used within a routine context, the authorization ID of the routine definer is to be used for authorization checking and for implicit qualification of unqualified object references within dynamic SQL statements within the routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES BIND.

INVOKERUN
If the package is used within a routine context, the current statement authorization ID in effect when the routine is invoked is to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES RUN.

INVOKEBIND
If the package is used within a routine context, the current statement authorization ID in effect when the routine is invoked is to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES BIND.

Because dynamic SQL statements will be using the authorization ID of the package owner in a package exhibiting bind behavior, the binder of the package should not have any authorities granted to them that the user of the package should not receive. Similarly, when defining a routine that will exhibit define behavior, the definer of the routine should not have any authorities granted to them that the user of the package should not receive since a dynamic statement will be using the authorization ID of the routine's definer.

The following dynamically prepared SQL statements cannot be used within a package that was not bound with DYNAMICRULES RUN: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY, and SET EVENT MONITOR STATE.

ENCODING
Specifies the encoding for all host variables in static statements in the plan or package. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

EXPLAIN
Stores information in the Explain tables about the access plans chosen for each SQL statement in the package. DRDA does not support the ALL value for this option.

NO Explain information will not be captured.

YES Explain tables will be populated with information about the chosen access plan at prep/bind time for static statements and at run time for incremental bind statements.

If the package is to be used for a routine and the package contains incremental bind statements, then the routine must be defined as MODIFIES SQL DATA. If this is not done, incremental bind statements in the package will cause a run time error (SQLSTATE 42985).

REOPT
Explain information for each reoptimizable incremental bind SQL statement is placed in the explain tables at run time. In addition, explain information is gathered for reoptimizable dynamic SQL statements at run time, even if the CURRENT EXPLAIN MODE register is set to NO.

If the package is to be used for a routine, the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

ONLY The ONLY option allows you to explain statements without having the privilege to execute them. The explain tables are populated but no persistent package is created. If an existing package with the same name and version is encountered during the bind process, the existing package is neither dropped nor replaced even if you specified ACTION REPLACE. If an error occurs during population of the explain tables, explain information is not added for the statement that returned the error and for any statements that follow it.

ALL Explain information for each eligible static SQL statement will be placed in the Explain tables at prep/bind time. Explain information for each eligible incremental bind SQL statement will be placed in the Explain tables at run time. In addition, Explain information will be gathered for eligible dynamic SQL statements at run time, even if the CURRENT EXPLAIN MODE register is set to `NO`.

If the package is to be used for a routine, the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985). This value for EXPLAIN is not supported by DRDA.

EXPLSNAP
Stores Explain Snapshot information in the Explain tables. This DB2 precompile/bind option is not supported by DRDA.

NO An Explain Snapshot will not be captured.

YES An Explain Snapshot for each eligible static SQL statement will be placed in the Explain tables at prep/bind time for static statements and at run time for incremental bind statements.

If the package is to be used for a routine and the package contains incremental bind statements, then the routine must be defined as MODIFIES SQL DATA or incremental bind statements in the package will cause a run time error (SQLSTATE 42985).

REOPT
Explain snapshot information for each reoptimizable incremental bind SQL statement is placed in the explain tables at run time. In addition, explain snapshot information is gathered for

reoptimizable dynamic SQL statements at run time, even if the CURRENT EXPLAIN SNAPSHOT register is set to NO.

If the package is to be used for a routine, the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

ALL An Explain Snapshot for each eligible static SQL statement will be placed in the Explain tables at prep/bind time. Explain snapshot information for each eligible incremental bind SQL statement will be placed in the Explain tables at run time. In addition, explain snapshot information will be gathered for eligible dynamic SQL statements at run time, even if the CURRENT EXPLAIN SNAPSHOT register is set to `NO`.

If the package is to be used for a routine, then the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

FEDERATED

Specifies whether a static SQL statement in a package references a nickname or a federated view. If this option is not specified and a static SQL statement in the package references a nickname or a federated view, a warning is returned and the package is created. This option is not supported for DRDA.

NO A nickname or federated view is not referenced in the static SQL statements of the package. If a nickname or federated view is encountered in a static SQL statement during the prepare or bind phase of this package, an error is returned and the package is *not* created.

YES A nickname or federated view can be referenced in the static SQL statements of the package. If no nicknames or federated views are encountered in static SQL statements during the prepare or bind of the package, no errors or warnings are returned and the package is created.

FEDERATED_ASYNCHRONY

Specifies the maximum number of asynchrony table queues (ATQs) that the federated server supports in the access plan for programs that use embedded SQL.

ANY 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.

number_of_atqs_in_the_plan

The number of ATQs in the plan. You specify a number in the range 0 to 32767.

FUNCPATH

Specifies the function path to be used in resolving user-defined distinct types and functions in static SQL. If this option is not specified, the default function path is "SYSIBM","SYSFUN",USER where USER is the value of the USER special register. This DB2 precompile/bind option is not supported by DRDA.

schema-name
An SQL identifier, either ordinary or delimited, which identifies a schema that exists at the application server. No validation that the schema exists is made at precompile or at bind time. The same schema cannot appear more than once in the function path. The schema name SYSPUBLIC cannot be specified for the function path. The number of schemas that can be specified is limited by the length of the resulting function path, which cannot exceed 2048 bytes. The schema SYSIBM does not need to be explicitly specified; it is implicitly assumed to be the first schema if it is not included in the function path.

GENERIC *string*
Supports the binding of new options that are defined in the target database, but are not supported by DRDA. Do not use this option to pass bind options that *are* defined in BIND or PRECOMPILE. This option can substantially improve dynamic SQL performance. The syntax is as follows:

```
generic "option1 value1 option2 value2 ..."
```

Each option and value must be separated by one or more blank spaces. For example, if the target DRDA database is DB2 Universal Database, Version 8, one could use:

```
generic "explsnap all queryopt 3 federated yes"
```

to bind each of the EXPLSNAP, QUERYOPT, and FEDERATED options.

The maximum length of the string is 32768 bytes.

GRANT

Note: If more than one of the GRANT, GRANT_GROUP, GRANT_USER, and GRANT_ROLE options are specified, only the last option specified is executed.

authid Grants EXECUTE and BIND privileges to a specified user name, role name or group ID. The SQL GRANT statement and its rules are used to determine the type of authid when none of USER, GROUP, or ROLE is provided to specify the type of the grantee on a GRANT statement. For the rules, see *GRANT (Role) statement*.

PUBLIC
Grants EXECUTE and BIND privileges to PUBLIC.

GRANT_GROUP *group-name*
Grants EXECUTE and BIND privileges to a specified group name.

GRANT_USER *user-name*
Grants EXECUTE and BIND privileges to a specified user name.

GRANT_ROLE *role-name*
Grants EXECUTE and BIND privileges to a specified role name.

INSERT
Allows a program being precompiled or bound against a DB2 Enterprise Server Edition server to request that data inserts be buffered to increase performance.

BUF Specifies that inserts from an application should be buffered.

DEF Specifies that inserts from an application should not be buffered.

ISOLATION
Determines how far a program bound to this package can be isolated from the effect of other executing programs.

CS Specifies Cursor Stability as the isolation level.

NC No Commit. Specifies that commitment control is not to be used. This isolation level is not supported by DB2 Database for Linux, UNIX, and Windows.

RR Specifies Repeatable Read as the isolation level.

RS Specifies Read Stability as the isolation level. Read Stability ensures that the execution of SQL statements in the package is isolated from other application processes for rows read and changed by the application.

UR Specifies Uncommitted Read as the isolation level.

IMMEDWRITE
Indicates whether immediate writes will be done for updates made to group buffer pool dependent pagesets or database partitions. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

KEEPDYNAMIC
Specifies whether dynamic SQL statements are to be kept after commit points. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

MESSAGES *message-file*
Specifies the destination for warning, error, and completion status messages. A message file is created whether the bind is successful or not. If a message file name is not specified, the messages are written to standard output. If the complete path to the file is not specified, the current directory is used. If the name of an existing file is specified, the contents of the file are overwritten.

OPTHINT
Controls whether query optimization hints are used for static SQL. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

OPTPROFILE *optimization-profile-name*
Specifies the name of an existing optimization profile to be used for all static statements in the package. The default value of the option is an empty string. The value also applies as the default for dynamic preparation of DML statements for which the CURRENT OPTIMIZATION PROFILE special register is null. If the specified name is unqualified, it is an SQL identifier, which is implicitly qualified by the QUALIFIER bind option.

The BIND command does not process the optimization file, but only validates that the name is syntactically valid. Therefore if the optimization profile does not exist or is invalid, an SQL0437W warning with reason code 13 will not occur until a DML statement is optimized using that optimization profile.

OS400NAMING
Specifies which naming option is to be used when accessing DB2 for System i data. Supported by DB2 for System i only. For a list of supported option values, refer to the documentation for DB2 for System i.

Because of the slashes used as separators, a DB2 utility can still report a syntax error at execution time on certain SQL statements which use the System i system naming convention, even though the utility might have been precompiled or bound with the OS400NAMING SYSTEM option. For example, the Command Line Processor will report a syntax error on an SQL CALL statement if the System i system naming convention is used, whether or not it has been precompiled or bound using the OS400NAMING SYSTEM option.

OWNER *authorization-id*
Designates a 128-byte authorization identifier for the package owner. The owner must have the privileges required to execute the SQL statements contained in the package. Only a user with DBADM authority can specify an authorization identifier other than the user ID. The default value is the authorization ID of the invoker of the precompile/bind process. SYSIBM, SYSCAT, and SYSSTAT are not valid values for this option. The *authorization-id* must be a user. A role or a group cannot be specified using the OWNER option.

PATH Specifies the function path to be used in resolving user-defined distinct types and functions in static SQL. If this option is not specified, the default function path is "SYSIBM","SYSFUN",USER where USER is the value of the USER special register.

schema-name
An SQL identifier, either ordinary or delimited, which identifies a schema that exists at the application server. No validation that the schema exists is made at precompile or at bind time.

QUALIFIER *qualifier-name*
Provides a 128-byte implicit qualifier for unqualified objects contained in the package. The default is the owner's authorization ID, whether or not **owner** is explicitly specified.

QUERYOPT *optimization-level*
Indicates the desired level of optimization for all static SQL statements contained in the package. The default value is 5. The SET CURRENT QUERY OPTIMIZATION statement describes the complete range of optimization levels available. This DB2 precompile/bind option is not supported by DRDA.

RELEASE
Indicates whether resources are released at each COMMIT point, or when the application terminates. This DRDA precompile/bind option is not supported by DB2 Database for Linux, UNIX, and Windows.

COMMIT
Release resources at each COMMIT point. Used for dynamic SQL statements.

DEALLOCATE
Release resources only when the application terminates.

SORTSEQ
Specifies which sort sequence table to use on System i. Supported by DB2 for System i only. For a list of supported option values, refer to the documentation for DB2 for System i.

SQLERROR
Indicates whether to create a package or a bind file if an error is encountered.

CHECK
Specifies that the target system performs all syntax and semantic checks on the SQL statements being bound. A package will not be created as part of this process. If, while binding, an existing package with the same name and version is encountered, the existing package is neither dropped nor replaced even if ACTION REPLACE was specified.

CONTINUE
Creates a package, even if errors occur when binding SQL statements. Those statements that failed to bind for authorization or existence reasons can be incrementally bound at execution time if VALIDATE RUN is also specified. Any attempt to execute them at run time generates an error (SQLCODE -525, SQLSTATE 51015).

NOPACKAGE
A package or a bind file is not created if an error is encountered.

REOPT
Specifies whether to have DB2 determine an access path at run time using values for host variables, parameter markers, global variables, and special registers. Valid values are:

NONE
The access path for a given SQL statement containing host variables, parameter markers, global variables, or special registers will not be optimized using real values. The default estimates for the these variables is used, and the plan is cached and will be used subsequently. This is the default value.

ONCE The access path for a given SQL statement will be optimized using the real values of the host variables, parameter markers, global variables, or special registers when the query is first executed. This plan is cached and used subsequently.

ALWAYS
The access path for a given SQL statement will always be compiled and reoptimized using the values of the host variables, parameter markers, global variables, or special registers that are known each time the query is executed.

REOPT | NOREOPT VARS
These options have been replaced by REOPT ALWAYS and REOPT NONE; however, they are still supported for previous compatibility. Specifies whether to have DB2 determine an access path at run time using values for host variables, global variables, parameter markers, and special registers. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

SQLWARN
Indicates whether warnings will be returned from the compilation of dynamic SQL statements (via PREPARE or EXECUTE IMMEDIATE), or from describe processing (via PREPARE...INTO or DESCRIBE).

NO Warnings will not be returned from the SQL compiler.

YES Warnings will be returned from the SQL compiler.

SQLCODE +236, +237 and +238 are exceptions. They are returned regardless of the SQLWARN option value.

STATICREADONLY

Determines whether static cursors will be treated as being READ ONLY. This DB2 precompile/bind option is not supported by DRDA.

NO All static cursors will take on the attributes as would normally be generated given the statement text and the setting of the LANGLEVEL precompile option. This is the default value.

YES Any static cursor that does not contain the FOR UPDATE or FOR READ ONLY clause will be considered READ ONLY.

STRDEL

Designates whether an apostrophe (') or double quotation marks (") will be used as the string delimiter within SQL statements. This DRDA precompile/bind option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

APOSTROPHE

Use an apostrophe (') as the string delimiter.

QUOTE

Use double quotation marks (") as the string delimiter.

TEXT *label*

The description of a package. Maximum length is 255 characters. The default value is blanks. This DRDA precompile/bind option is not supported by DB2 Database for Linux, UNIX, and Windows.

TRANSFORM GROUP

Specifies the transform group name to be used by static SQL statements for exchanging user-defined structured type values with host programs. This transform group is not used for dynamic SQL statements or for the exchange of parameters and results with external functions or methods. This option is not supported by DRDA.

groupname

An SQL identifier of up to 18 bytes in length. A group name cannot include a qualifier prefix and cannot begin with the prefix SYS since this is reserved for database use. In a static SQL statement that interacts with host variables, the name of the transform group to be used for exchanging values of a structured type is as follows:

- The group name in the TRANSFORM GROUP bind option, if any
- The group name in the TRANSFORM GROUP prep option as specified at the original precompilation time, if any
- The DB2_PROGRAM group, if a transform exists for the given type whose group name is DB2_PROGRAM
- No transform group is used if none of the above conditions exist.

The following errors are possible during the bind of a static SQL statement:

- SQLCODE yyyyy, SQLSTATE xxxxx: A transform is needed, but no static transform group has been selected.

- SQLCODE yyyyy, SQLSTATE xxxxx: The selected transform group does not include a necessary transform (TO SQL for input variables, FROM SQL for output variables) for the data type that needs to be exchanged.
- SQLCODE yyyyy, SQLSTATE xxxxx: The result type of the FROM SQL transform is not compatible with the type of the output variable, or the parameter type of the TO SQL transform is not compatible with the type of the input variable.

In these error messages, *yyyyy* is replaced by the SQL error code, and *xxxxx* by the SQL state code.

VALIDATE

Determines when the database manager checks for authorization errors and object not found errors. The package owner authorization ID is used for validity checking.

BIND Validation is performed at precompile/bind time. If all objects do not exist, or all authority is not held, error messages are produced. If SQLERROR CONTINUE is specified, a package/bind file is produced despite the error message, but the statements in error are not executable.

RUN Validation is attempted at bind time. If all objects exist, and all authority is held, no further checking is performed at execution time.

If all objects do not exist, or all authority is not held at precompile/bind time, warning messages are produced, and the package is successfully bound, regardless of the SQLERROR CONTINUE option setting. However, authority checking and existence checking for SQL statements that failed these checks during the precompile/bind process can be redone at execution time.

Examples

The following example binds `myapp.bnd` (the bind file generated when the `myapp.sqc` program was precompiled) to the database to which a connection has been established:

```
db2 bind myapp.bnd
```

Any messages resulting from the bind process are sent to standard output.

Usage notes

Binding a package using the REOPT option with the ONCE or ALWAYS value specified might change the static and dynamic statement compilation and performance.

Binding can be done as part of the precompile process for an application program source file, or as a separate step at a later time. Use BIND when binding is performed as a separate process.

The name used to create the package is stored in the bind file, and is based on the source file name from which it was generated (existing paths or extensions are discarded). For example, a precompiled source file called `myapp.sql` generates a default bind file called `myapp.bnd` and a default package name of `MYAPP`. However,

the bind file name and the package name can be overridden at precompile time by using the BINDFILE and the PACKAGE options.

Binding a package with a schema name that does not already exist results in the implicit creation of that schema. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.

BIND executes under the transaction that was started. After performing the bind, BIND issues a COMMIT or a ROLLBACK to terminate the current transaction and start another one.

Binding stops if a fatal error or more than 100 errors occur. If a fatal error occurs, the utility stops binding, attempts to close all files, and discards the package.

When a package exhibits bind behavior, the following will be true:

1. The implicit or explicit value of the BIND option OWNER will be used for authorization checking of dynamic SQL statements.
2. The implicit or explicit value of the BIND option QUALIFIER will be used as the implicit qualifier for qualification of unqualified objects within dynamic SQL statements.
3. The value of the special register CURRENT SCHEMA has no effect on qualification.

In the event that multiple packages are referenced during a single connection, all dynamic SQL statements prepared by those packages will exhibit the behavior as specified by the DYNAMICRULES option for that specific package and the environment they are used in.

Parameters displayed in the SQL0020W message are correctly noted as errors, and will be ignored as indicated by the message.

If an SQL statement is found to be in error and the BIND option SQLERROR CONTINUE was specified, the statement will be marked as invalid. In order to change the state of the SQL statement, another BIND must be issued . Implicit and explicit rebind will not change the state of an invalid statement. In a package bound with VALIDATE RUN, a statement can change from static to incremental bind or incremental bind to static across implicit and explicit rebinds depending on whether or not object existence or authority problems exist during the rebind.

The privileges from the roles granted to the authorization identifier used to bind the package (the value of the OWNER bind option) or to PUBLIC, are taken into account when binding a package. Roles acquired through groups, in which the authorization identifier used to bind the package is a member, will not be used.

For an embedded SQL program, if the bind option is not explicitly specified the static statements in the package are bound using the FEDERATED_ASYNC configuration parameter. If the FEDERATED_ASYNCHRONY bind option is specified explicitly, that value is used for binding the packages and is also the initial value of the special register. Otherwise, the value of the database manager configuration parameter is used as the initial value of the special register. The FEDERATED_ASYNCHRONY bind option influences dynamic SQL only when it is explicitly set.

The value of the FEDERATED_ASYNCHRONY bind option is recorded in the FEDERATED_ASYNCHRONY column in the SYSCAT.PACKAGES catalog table.

When the bind option is not explicitly specified, the value of FEDERATED_ASYNC configuration parameter is used and the catalog shows a value of -2 for the FEDERATED_ASYNCHRONY column.

If the FEDERATED_ASYNCHRONY bind option is not explicitly specified when a package is bound, and if this package is implicitly or explicitly rebound, the package is rebound using the current value of the FEDERATED_ASYNC configuration parameter.

CATALOG DATABASE

Stores database location information in the system database directory. The database can be located either on the local workstation or on a remote database partition server.

Scope

In a partitioned database environment, when cataloging a local database into the system database directory, this command must be issued from a database partition on the server where the database resides.

Authorization

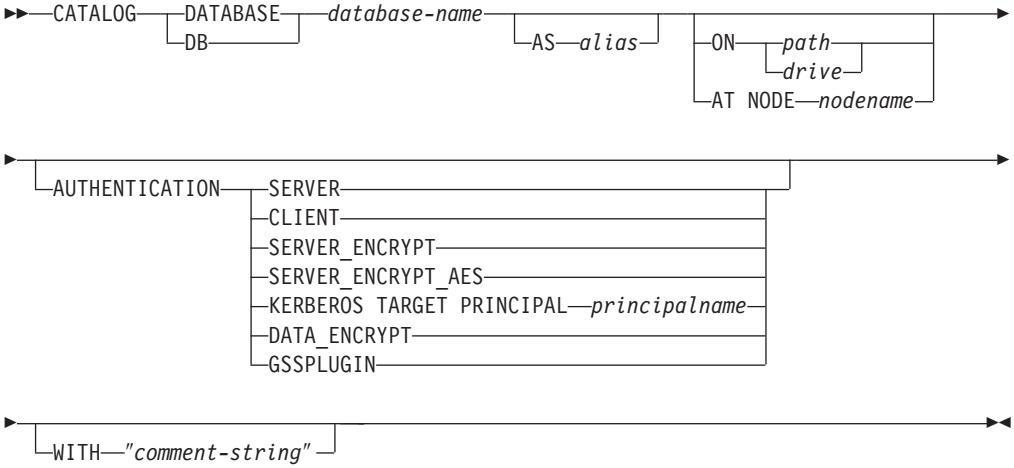
One of the following:

- SYSADM
- SYSCTRL

Required connection

None. Directory operations affect the local directory only.

Command syntax



Command parameters

DATABASE *database-name*
Specifies the name of the database to catalog.

AS *alias*

Specifies an alias as an alternate name for the database being cataloged. If an alias is not specified, the database manager uses *database-name* as the alias.

ON *path* | *drive*

Specifies the path on which the database being cataloged resides. On Windows operating systems, may instead specify the letter of the drive on which the database being cataloged resides (if it was created on a drive, not on a specific path).

AT NODE *nodename*

Specifies the name of the database partition server where the database being cataloged resides. This name should match the name of an entry in the node directory. If the node name specified does not exist in the node directory, a warning is returned, but the database is cataloged in the system database directory. The node name should be cataloged in the node directory if a connection to the cataloged database is desired.

AUTHENTICATION

The authentication value is stored for remote databases (it appears in the output from the LIST DATABASE DIRECTORY command) but it is not stored for local databases.

Specifying an authentication type can result in a performance benefit.

SERVER

Specifies that authentication takes place on the database partition server containing the target database.

CLIENT

Specifies that authentication takes place on the database partition server where the application is invoked.

SERVER_ENCRYPT

Specifies that authentication takes place on the database partition server containing the target database, and that user IDs and passwords are encrypted at the source. User IDs and passwords are decrypted at the target, as specified by the authentication type cataloged at the source.

KERBEROS

Specifies that authentication takes place using Kerberos Security Mechanism.

TARGET PRINCIPAL *principalname*

Fully qualified Kerberos principal name for the target server; that is, the fully qualified Kerberos principal of the DB2 instance owner in the form of `name/instance@REALM`. For Windows 2000, Windows XP, and Windows Server 2003, this is the logon account of the DB2 server service in the form of *`userid@DOMAIN`*, *`userid@xxx.xxx.xxx.com`* or *`domain\userid`*.

DATA_ENCRYPT

Specifies that authentication takes place on the database partition server containing the target database, and that connections must use data encryption.

GSSPLUGIN
Specifies that authentication takes place using an external GSS API-based plug-in security mechanism.

SERVER_ENCRYPT_AES
Specifies that authentication takes place on the database partition server containing the target database, and that user IDs and passwords are encrypted with an Advanced Encryption Standard (AES) encryption algorithm at the source and decrypted at the target.

WITH *"comment-string"*
Describes the database or the database entry in the system database directory. The maximum length of a comment string is 30 characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by double quotation marks.

Examples

```
db2 catalog database sample on /databases/sample
   with "Sample Database"
```

Usage notes

Use CATALOG DATABASE to catalog databases located on local or remote database partition servers, recatalog databases that were uncataloged previously, or maintain multiple aliases for one database (regardless of database location).

DB2 automatically catalogs databases when they are created. It catalogs an entry for the database in the local database directory and another entry in the system database directory. If the database is created from a remote client (or a client which is executing from a different instance on the same machine), an entry is also made in the system database directory at the client instance.

If neither path nor database partition server name is specified, the database is assumed to be local, and the location of the database is assumed to be that specified in the database manager configuration parameter **dftdbpath**.

Databases on the same database partition server as the database manager instance are cataloged as *indirect* entries. Databases on other database partition servers are cataloged as *remote* entries.

CATALOG DATABASE automatically creates a system database directory if one does not exist. The system database directory is stored on the path that contains the database manager instance that is being used, and is maintained outside of the database.

List the contents of the system database directory using the LIST DATABASE DIRECTORY command. To list the contents of the local database directory use the LIST DATABASE DIRECTORY **ON** *path*, where *path* is where the database was created.

If directory caching is enabled, database, node and DCS directory files are cached in memory. To see if directory caching is enabled, check the value for the *dir_cache* directory cache support configuration parameter in the output from the GET DATABASE MANAGER CONFIGURATION command. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed

when the application modifies any of the directory files, directory changes made by other applications might not be effective until the application has restarted.

To refresh the CLP's directory cache, use the TERMINATE command. To refresh the database manager's shared cache, stop (db2stop) and then restart (db2start) the database manager. To refresh the directory cache for another application, stop and then restart that application.

CREATE DATABASE

The CREATE DATABASE command initializes a new database with an optional user-defined collating sequence, creates the three initial table spaces, creates the system tables, and allocates the recovery log file. When you initialize a new database, the AUTOCONFIGURE command is issued by default.

Note: When the instance and database directories are created by the DB2 database manager, the permissions are accurate and should not be changed.

When the CREATE DATABASE command is issued, the Configuration Advisor also runs automatically. This means that the database configuration parameters are automatically tuned for you according to your system resources. In addition, Automated Runstats is enabled. To disable the Configuration Advisor from running at database creation, refer to the *db2_enable_autoconfig_default* registry variable. To disable Automated Runstats, refer to *auto_runstats* database configuration parameter.

Adaptive Self Tuning Memory is also enabled by default for single partition databases. To disable Adaptive Self Tuning Memory by default, refer to the *self_tuning_mem* database configuration parameter (see *self_tuning_mem - Self-tuning memory configuration parameter*). For multi-partition databases, Adaptive Self Tuning Memory is disabled by default.

If no code set is specified on the CREATE DATABASE command, then the collations allowed are: SYSTEM, IDENTITY_16BIT, UCA400_NO, UCA400_LSK, UCA400_LTH, *language-aware-collation*, and *locale-aware-collation* (SQLCODE -1083). The default code set for a database is UTF-8. If a particular code set and territory is needed for a database, the desired code set and territory should be specified in the CREATE DATABASE command.

This command is not valid on a client.

Scope

In a partitioned database environment, this command affects all database partitions that are listed in the db2nodes.cfg file.

The database partition from which this command is issued becomes the catalog database partition for the new database.

Authorization

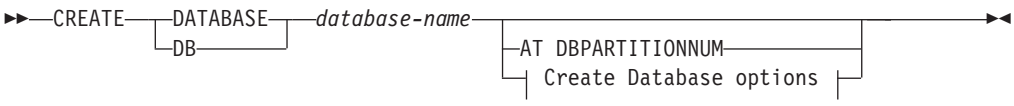
You must have one of the following:

- *sysadm*
- *sysctrl*

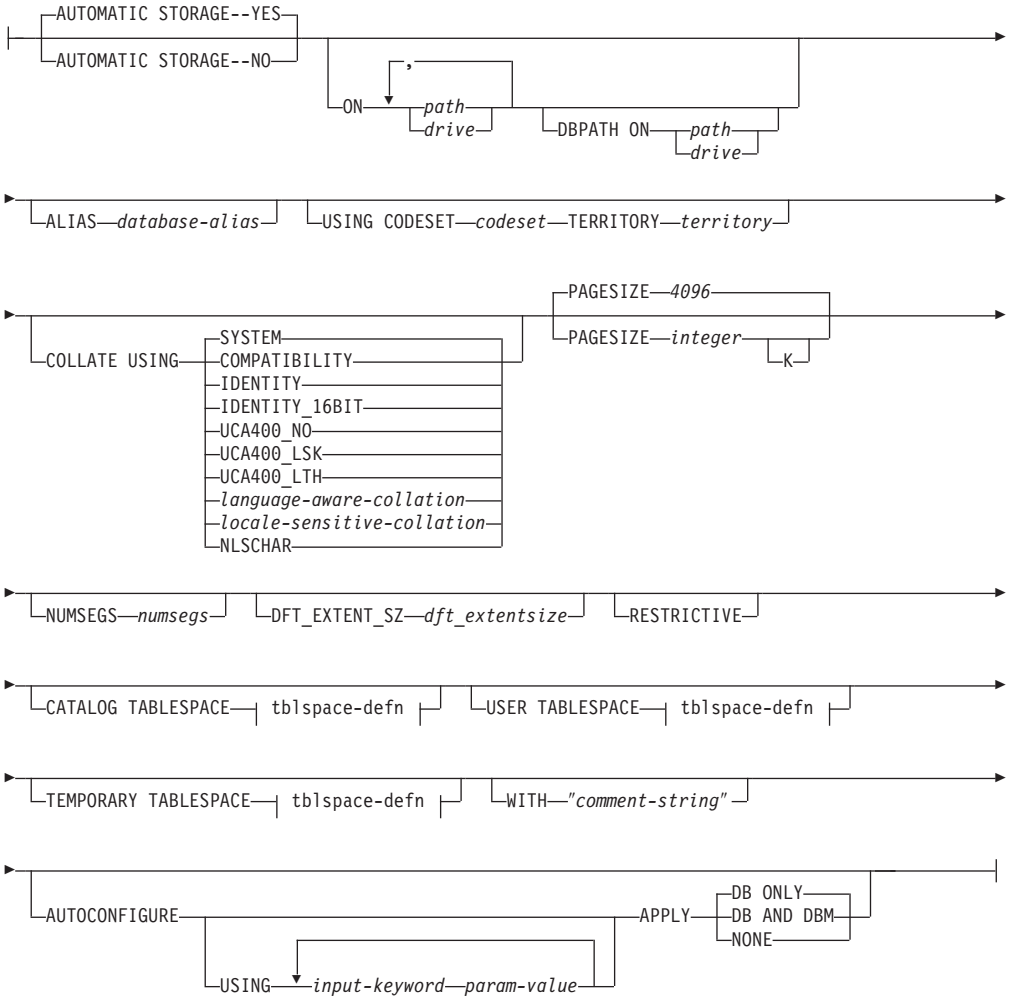
Required connection

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 this command during processing.

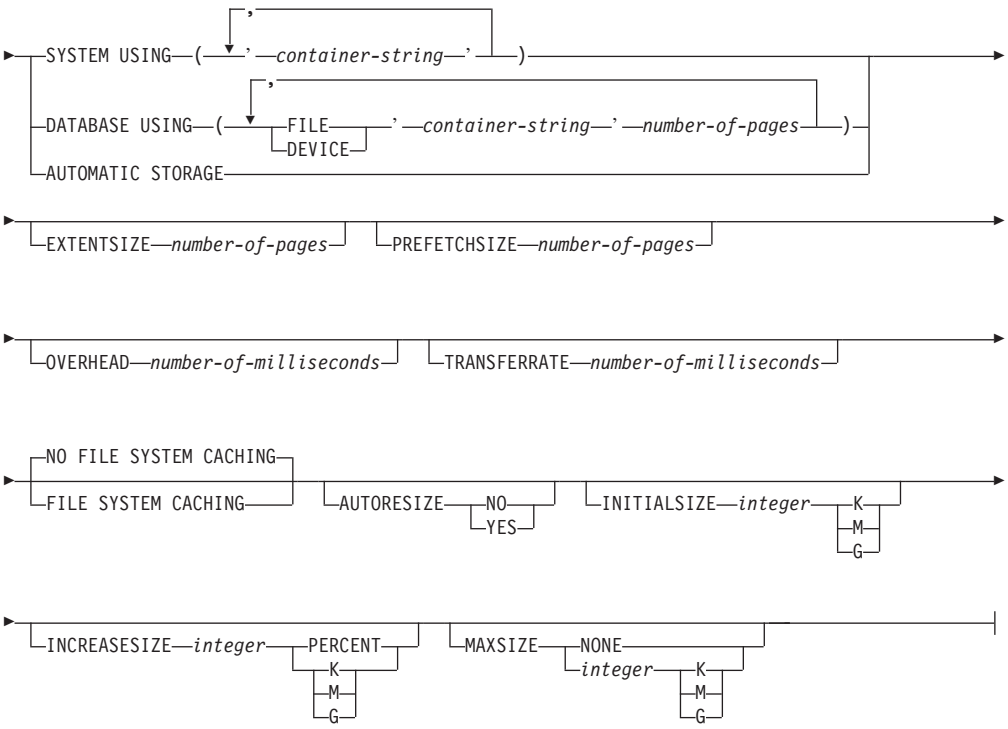
Command syntax



Create Database options:



tblspace-defn:



Note:

1. The combination of the code set and territory values must be valid.
2. Not all collating sequences are valid with every code set and territory combination.
3. The table space definitions specified on CREATE DATABASE apply to all database partitions on which the database is being created. They cannot be specified separately for each database partition. If the table space definitions are to be created differently on particular database partitions, the CREATE TABLESPACE statement must be used.

   When defining containers for table spaces, $N can be used. $N will be replaced by the database partition number when the container is actually created. This is required if the user wants to specify containers in a multiple logical partition database.
4. The AUTOCONFIGURE option requires *sysadm* authority.

Command parameters

DATABASE *database-name*

A name to be assigned to the new database. This must be a unique name that differentiates the database from any other database in either the local database directory or the system database directory. The name must conform to naming conventions for databases. Specifically, the name must not contain any space characters.

AT DBPARTITIONNUM

Specifies that the database is to be created only on the database partition that issues the command. You do not specify this option when you create a new database. You can use it to recreate a database partition that you dropped because it was damaged. After you use the CREATE DATABASE

command with the AT DBPARTITIONNUM option, the database at this database partition is in the restore-pending state. You must immediately restore the database on this database partition server. This parameter is not intended for general use. For example, it should be used with RESTORE DATABASE command if the database partition at a database partition server was damaged and must be recreated. Improper use of this parameter can cause inconsistencies in the system, so it should only be used with caution.

If this parameter is used to recreate a database partition that was dropped (because it was damaged), the database at this database partition will be in the restore-pending state. After recreating the database partition, the database must immediately be restored on this database partition.

AUTOMATIC STORAGE NO | YES
Specifies that automatic storage is being explicitly disabled or enabled for the database. The default value is YES. If the AUTOMATIC STORAGE clause is not specified, automatic storage is implicitly enabled by default.

NO Automatic storage is not being enabled for the database.

YES Automatic storage is being enabled for the database.

ON *path* **or** *drive*
The meaning of this option depends on the value of the AUTOMATIC STORAGE option.

- If AUTOMATIC STORAGE NO is specified, automatic storage is disabled for the database. In this case, only one path can be included as part of the ON option, and it specifies the path on which to create the database. If a path is not specified, the database is created on the default database path that is specified in the database manager configuration file (**dftdbpath** parameter). This behavior matches that of DB2 Universal Database Version 8.2 and earlier.
- Otherwise, automatic storage is enabled for the database by default. In this case, multiple paths may be listed here, each separated by a comma. These are referred to as storage paths and are used to hold table space containers for automatic storage table spaces. For multi-partition databases the same storage paths will be used on all partitions.

  The DBPATH ON option specifies on which paths to create the database. If the DBPATH ON option is not specified, the database is created on the first path listed in the ON option. If no paths are specified with the ON option, the database is created on the default database path that is specified in the database manager configuration file (**dftdbpath** parameter). This will also be used as the location for the single storage path associated with the database.

  The database path is the location where a hierarchical directory structure is created. The structure holds the following files needed for the operation of the database:

  – Buffer pool information
  – Table space information
  – Storage path information
  – Database configuration information
  – History file information regarding backups, restores, loading of tables, reorganization of tables, altering of table spaces, and other database changes
  – Log control files with information about active logs

The DBPATH ON option can be used to place these files and information in a directory that is separate from the storage paths where the database data is kept. It is suggested that the DBPATH ON option be used when automatic storage is enabled to keep the database information separate from the database data.

The maximum length of a path is 175 characters.

For MPP systems, a database should not be created in an NFS-mounted directory. If a path is not specified, ensure that the *dftdbpath* database manager configuration parameter is not set to an NFS-mounted path (for example, on UNIX based systems, it should not specify the `$HOME` directory of the instance owner). The path specified for this command in an MPP system cannot be a relative path. Also, all paths specified as part of the ON option must exist on all database partitions.

A given database path or storage path must exist and be accessible on each database partition.

DBPATH ON *path* **or** *drive*
If automatic storage is enabled, the DBPATH ON option specifies the path on which to create the database. If automatic storage is enabled and the DBPATH ON option is not specified, the database is created on the first path listed with the ON option.

The maximum length of a database path is 215 characters and the maximum length of a storage path is 175 characters.

ALIAS *database-alias*
An alias for the database in the system database directory. If no alias is provided, the specified database name is used.

USING CODESET *codeset*
Specifies the code set to be used for data entered into this database. After you create the database, you cannot change the specified code set.

TERRITORY *territory*
Specifies the territory identifier or locale identifier to be used for data entered into this database. After you create the database, you cannot change the specified territory. The combination of the code set and territory or locale values must be valid.

COLLATE USING

Identifies the type of collating sequence to be used for the database. Once the database has been created, the collating sequence cannot be changed.

In a Unicode database, the catalog tables and views are always created with the IDENTITY collation, regardless of the collation specified in the COLLATE USING clause. In non-Unicode databases, the catalog tables and views are created with the database collation.

COMPATIBILITY
The DB2 Version 2 collating sequence. Some collation tables have been enhanced. This option specifies that the previous version of these tables is to be used.

IDENTITY
Identity collating sequence, in which strings are compared byte for byte. This is the default for Unicode databases.

IDENTITY_16BIT
CESU-8 (Compatibility Encoding Scheme for UTF-16: 8-Bit)

collation sequence as specified by the Unicode Technical Report #26, which is available at the Unicode Consortium Web site (www.unicode.org). This option can only be specified when creating a Unicode database.

UCA400_NO
The UCA (Unicode Collation Algorithm) collation sequence that is based on the Unicode Standard version 4.0.0 with normalization implicitly set to ON. Details of the UCA can be found in the Unicode Technical Standard #10, which is available at the Unicode Consortium Web site (www.unicode.org). This option can only be used when creating a Unicode database.

UCA400_LSK
The UCA (Unicode Collation Algorithm) collation sequence based on the Unicode Standard version 4.0.0 but will sort Slovak characters in the appropriate order. Details of the UCA can be found in the Unicode Technical Standard #10, which is available at the Unicode Consortium Web site (www.unicode.org). This option can only be used when creating a Unicode database.

UCA400_LTH
The UCA (Unicode Collation Algorithm) collation sequence that is based on the Unicode Standard version 4.0.0 but will sort all Thai characters according to the Royal Thai Dictionary order. Details of the UCA can be found in the Unicode Technical Standard #10 available at the Unicode Consortium Web site (www.unicode.org). This option can only be used when creating a Unicode database. This collator might order Thai data differently from the NLSCHAR collator option.

language-aware-collation
This option can only be used for Unicode databases. The database collating sequence is based on the SYSTEM collation for a non-Unicode database. This string must be of the format SYSTEM_*codepage_territory*. If the string supplied is invalid, the create database will fail (SQLCODE -204; object not found). See *Language-aware collations for Unicode data* for more information and for the naming of system based collations.

Note: When the CREATE DATABASE command is performed against a Version 9.0 server, this option cannot be used. By default, a Unicode database on such a server will be created with SYSTEM collation.

locale-sensitive-collation
This option can only be used for Unicode databases. See *Unicode Collation Algorithm based collations* for more information and for the naming of locale-sensitive UCA-based collations. If the collation name provided is invalid, the CREATE DATABASE command execution will fail (SQLCODE -204).

NLSCHAR
System-defined collating sequence using the unique collation rules for the specific code set/territory.

This option can only be used with the Thai code page (CP874). If this option is specified in non-Thai environments, the command will fail and return the error SQL1083N with Reason Code 4.

SYSTEM
This is the default option when creating a database. For non-Unicode databases, the collating sequence is based on the database territory. For Unicode databases, this option maps to a language-aware collation, based on the client code set and territory. If an appropriate language-aware collation is unavailable, then the IDENTITY collation is used.

PAGESIZE *integer*
Specifies the page size of the default buffer pool along with the initial table spaces (SYSCATSPACE, TEMPSPACE1, USERSPACE1) when the database is created. This also represents the default page size for all future CREATE BUFFERPOOL and CREATE TABLESPACE statements. The valid values for integer without the suffix K are 4 096, 8 192, 16 384, or 32 768. The valid values for integer with the suffix K are 4, 8, 16, or 32. At least one space is required between the integer and the suffix K. The default is a page size of 4 096 bytes (4 K).

NUMSEGS *numsegs*
Specifies the number of directories (table space containers) that will be created and used to store the database table files for any default SMS table spaces. This parameter does not affect automatic storage table spaces, DMS table spaces, any SMS table spaces with explicit creation characteristics (created when the database is created), or any SMS table spaces explicitly created after the database is created.

DFT_EXTENT_SZ *dft_extentsize*
Specifies the default extent size of table spaces in the database.

RESTRICTIVE
If the RESTRICTIVE option is present it causes the RESTRICT_ACCESS database configuration parameter to be set to YES and no privileges or authorities are automatically granted to PUBLIC. If the RESTRICTIVE option is not present then the RESTRICT_ACCESS database configuration parameter is set to NO and all of the following privileges are automatically granted to PUBLIC.

- BIND on all packages created in the NULLID schema
- BINDADD
- CONNECT
- CREATEIN on schema SQLJ
- CREATEIN on schema NULLID
- CREATETAB
- EXECUTE with GRANT on all procedures in schema SQLJ
- EXECUTE with GRANT on all functions and procedures in schema SYSFUN
- EXECUTE with GRANT on all functions and procedures in schema SYSPROC (except audit routines)
- EXECUTE on all table functions in schema SYSIBM
- EXECUTE on all other procedures in schema SYSIBM
- EXECUTE on all packages created in the NULLID schema
- IMPLICIT_SCHEMA
- SELECT access to the SYSIBM catalog tables
- SELECT access to the SYSCAT catalog views
- SELECT access to the SYSSTAT catalog views

- UPDATE access to the SYSSTAT catalog views
- USAGE on the SYSDEFAULTUSERWORKLOAD workload
- USE on table space USERSPACE1

CATALOG TABLESPACE tblspace-defn

Specifies the definition of the table space that will hold the catalog tables, SYSCATSPACE. If not specified and automatic storage is not enabled for the database, SYSCATSPACE is created as a System Managed Space (SMS) table space with *numsegs* number of directories as containers, and with an extent size of *dft_extentsize*. For example, the following containers would be created if *numsegs* were specified to be 5:

```
/u/smith/smith/NODE0000/SQL00001/SQLT0000.0
/u/smith/smith/NODE0000/SQL00001/SQLT0000.1
/u/smith/smith/NODE0000/SQL00001/SQLT0000.2
/u/smith/smith/NODE0000/SQL00001/SQLT0000.3
/u/smith/smith/NODE0000/SQL00001/SQLT0000.4
```

If not specified and automatic storage is enabled for the database, SYSCATSPACE is created as an automatic storage table space with its containers created on the defined storage paths. The extent size of this table space is 4. Appropriate values for AUTORESIZE, INITIALSIZE, INCREASESIZE, and MAXSIZE are set automatically.

See *CREATE TABLESPACE statement* for more information on the table space definition fields.

In a partitioned database environment, the catalog table space is only created on the catalog database partition, the database partition on which the CREATE DATABASE command is issued.

USER TABLESPACE tblspace-defn

Specifies the definition of the initial user table space, USERSPACE1. If not specified and automatic storage is not enabled for the database, USERSPACE1 is created as an SMS table space with *numsegs* number of directories as containers and with an extent size of *dft_extentsize*. For example, the following containers would be created if *numsegs* were specified to be 5:

```
/u/smith/smith/NODE0000/SQL00001/SQLT0001.0
/u/smith/smith/NODE0000/SQL00001/SQLT0002.1
/u/smith/smith/NODE0000/SQL00001/SQLT0002.2
/u/smith/smith/NODE0000/SQL00001/SQLT0002.3
/u/smith/smith/NODE0000/SQL00001/SQLT0002.4
```

If not specified and automatic storage is enabled for the database, USERSPACE1 is created as an automatic storage table space with its containers created on the defined storage paths. The extent size of this table space will be *dft_extentsize*. Appropriate values for AUTORESIZE, INITIALSIZE, INCREASESIZE, and MAXSIZE are set automatically.

See *CREATE TABLESPACE statement* for more information on the table space definition fields.

TEMPORARY TABLESPACE tblspace-defn

Specifies the definition of the initial system temporary table space, TEMPSPACE1. If not specified and automatic storage is not enabled for the database, TEMPSPACE1 is created as an SMS table space with *numsegs* number of directories as containers and with an extent size of *dft_extentsize*. For example, the following containers would be created if *numsegs* were specified to be 5:

```
/u/smith/smith/NODE0000/SQL00001/SQLT0002.0
/u/smith/smith/NODE0000/SQL00001/SQLT0001.1
/u/smith/smith/NODE0000/SQL00001/SQLT0001.2
/u/smith/smith/NODE0000/SQL00001/SQLT0001.3
/u/smith/smith/NODE0000/SQL00001/SQLT0001.4
```

If not specified and automatic storage is enabled for the database, TEMPSPACE1 is created as an automatic storage table space with its containers created on the defined storage paths. The extent size of this table space is *dft_extentsize*.

See *CREATE TABLESPACE statement* for more information on the table space definition fields.

tblspace-defn

Various table space definitions can be specified through the following command parameters.

MANAGED BY

SYSTEM USING *container-string*

Specifies that the table space is to be an SMS table space. When the type of table space is not specified, the default behavior is to create a regular table space.

For an SMS table space, identifies one or more containers that will belong to the table space and in which the table space data will be stored. The *container-string* cannot exceed 240 bytes in length.

Each *container-string* can be an absolute or relative directory name.

The directory name, if not absolute, is relative to the database directory, and can be a path name alias (a symbolic link on UNIX systems) to storage that is not physically associated with the database directory. For example, *dbdir*`/work/c1` could be a symbolic link to a separate file system.

If any component of the directory name does not exist, it is created by the database manager. When a table space is dropped, all components created by the database manager are deleted. If the directory identified by *container-string* exists, it must not contain any files or subdirectories (SQLSTATE 428B2).

The format of *container-string* is dependent on the operating system. On Windows operating systems, an absolute directory path name begins with a drive letter and a colon (:); on UNIX systems, an absolute path name begins with a forward slash (/). A relative path name on any platform does not begin with an operating system-dependent character.

Remote resources (such as LAN-redirected drives or NFS-mounted file systems) are currently only supported when using Network Appliance Filers, IBM iSCSI, IBM Network Attached Storage, Network Appliance iSCSI, NEC iStorage S2100, S2200, or S4100, or NEC Storage NS Series with a Windows DB2 server. Note that NEC Storage NS Series is only supported with the use of an uninterrupted

power supply (UPS); continuous UPS (rather than standby) is recommended. An NFS-mounted file system on AIX must be mounted in uninterruptible mode using the -o nointr option.

DATABASE USING
Specifies that the table space is to be a DMS table space. When the type of table space is not specified, the default behavior is to create a large table space.

For a DMS table space, identifies one or more containers that will belong to the table space and in which the table space data will be stored. The type of the container (either FILE or DEVICE) and its size (in PAGESIZE pages) are specified. A mixture of FILE and DEVICE containers can be specified. The *container-string* cannot exceed 254 bytes in length.

Remote resources (such as LAN-redirected drives or NFS-mounted file systems) are currently only supported when using Network Appliance Filers, IBM iSCSI, IBM Network Attached Storage, Network Appliance iSCSI, NEC iStorage S2100, S2200, or S4100, or NEC Storage NS Series with a Windows DB2 server. Note that NEC Storage NS Series is only supported with the use of an uninterrupted power supply (UPS); continuous UPS (rather than standby) is recommended..

All containers must be unique across all databases. A container can belong to only one table space. The size of the containers can differ; however, optimal performance is achieved when all containers are the same size. The exact format of *container-string* is dependent on the operating system.

FILE *container-string number-of-pages*

For a FILE container, *container-string* must be an absolute or relative file name. The file name, if not absolute, is relative to the database directory. If any component of the directory name does not exist, it is created by the database manager. If the file does not exist, it will be created and initialized to the specified size by the database manager. When a table space is dropped, all components created by the database manager are deleted.

Note: If the file exists, it is overwritten, and if it is smaller than specified, it is extended. The file will not be truncated if it is larger than specified.

DEVICE *container-string number-of-pages*
For a DEVICE container, *container-string* must be a device name. The device must already exist.

AUTOMATIC STORAGE
Specifies that the table space is to be an automatic storage table space. If automatic storage is not defined for the database, an error is returned (SQLSTATE 55060).

An automatic storage table space is created as either a system managed space (SMS) table space or a database managed space (DMS) table space. When DMS is chosen and the type of table space is not specified, the default behavior is to create a large table space. With an automatic storage table space, the database manager determines which containers are to be assigned to the table space, based upon the storage paths that are associated with the database.

EXTENTSIZE *number-of-pages*
Specifies the number of PAGESIZE pages that will be written to a container before skipping to the next container. The extent size value can also be specified as an integer value followed by K (for kilobytes) or M (for megabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the value for the extent size. The database manager cycles repeatedly through the containers as data is stored.

The default value is provided by the DFT_EXTENT_SZ database configuration parameter, which has a valid range of 2-256 pages.

PREFETCHSIZE *number-of-pages*
Specifies the number of PAGESIZE pages that will be read from the table space when data prefetching is being performed. The prefetch size value can also be specified as an integer value followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the number of pages value for prefetch size.

OVERHEAD *number-of-milliseconds*
Specifies the I/O controller overhead and disk seek and latency time. This value is used to determine the cost of I/O during query optimization. The value of *number-of-milliseconds* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all containers, the number should be the average for all containers that belong to the table space.

For a database that was created in Version 9 or later, the default I/O controller overhead and disk seek and latency time is 7.5 milliseconds. For a database that was upgraded from a previous version of DB2 to Version 9 or later, the default is 12.67 milliseconds.

TRANSFERRATE *number-of-milliseconds*
Specifies the time to read one page into memory. This value is used to determine the cost of I/O during query optimization. The value of *number-of-milliseconds* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all containers, the number should be the average for all containers that belong to the table space.

For a database that was created in Version 9 or later, the default time to read one page into memory is 0.06 milliseconds. For a database that was upgraded from a previous version of DB2 to Version 9 or later, the default is 0.18 milliseconds.

NO FILE SYSTEM CACHING
Specifies that all I/O operations are to bypass the file system-level cache. See *Table spaces without file system caching* for more details. This is the default option on most configurations. See *File system caching configurations* for details.

FILE SYSTEM CACHING
Specifies that all I/O operations in the target table space are to be cached at the file system level. See *Table spaces without file system caching* for more details. This is the default option on some configurations. See *File system caching configurations* for details.

AUTORESIZE
Specifies whether or not the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled. Auto-resizable table spaces automatically increase in size when they become full. The default is NO for DMS table spaces and YES for automatic storage table spaces.

NO Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be disabled.

YES Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled.

INITIALSIZE *integer*
Specifies the initial size, per database partition, of an automatic storage table space. This option is only valid for automatic storage table spaces. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain a consistent size across containers in the table space. Moreover, if the table space is auto-resizable and the initial size is not large enough to contain meta-data that must be added to the new table space, the database manager will continue to extend the table space by the value of **INCREASESIZE** until there is enough space. If the **INITIALSIZE** clause is not specified, the database manager determines an appropriate value. The value for *integer* must be at least 48 K.

K K (for kilobytes).

M M (for megabytes).

G G (for gigabytes).

INCREASESIZE *integer*

Specifies the amount, per database partition, by which a table space that is enabled for auto-resize will automatically be increased when the table space is full, and a request for space has been made. The integer value must be followed by either:

- PERCENT to specify the amount as a percentage of the table space size at the time that a request for space is made. When PERCENT is specified, the integer value must be between 0 and 100 (SQLSTATE 42615).
- K (for kilobytes), M (for megabytes), or G (for gigabytes) to specify the amount in bytes

Note that the actual value used might be slightly smaller or larger than what was specified, because the database manager strives to maintain consistent growth across containers in the table space. If the table space is auto-resizable, but the INCREASESIZE clause is not specified, the database manager determines an appropriate value.

PERCENT

Percent from 0 to 100.

K K (for kilobytes).

M M (for megabytes).

G G (for gigabytes).

MAXSIZE

Specifies the maximum size to which a table space that is enabled for auto-resize can automatically be increased. If the table space is auto-resizable, but the MAXSIZE clause is not specified, the default is NONE.

NONE

Specifies that the table space is to be allowed to grow to file system capacity, or to the maximum table space size.

integer Specifies a hard limit on the size, per database partition, to which a DMS table space or an automatic storage table space can automatically be increased. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain consistent growth across containers in the table space.

K K (for kilobytes).

M M (for megabytes).

G G (for gigabytes).

WITH *comment-string*

Describes the database entry in the database directory. Any comment that helps to describe the database can be entered. Maximum length is 30

characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by single or double quotation marks.

AUTOCONFIGURE

Based on user input, calculates the recommended settings for buffer pool size, database configuration, and database manager configuration and optionally applies them. The Configuration Advisor is run by default when the CREATE DATABASE command is issued. The AUTOCONFIGURE option is needed only if you want to tweaks the recommendations.

USING *input-keyword param-value*

Table 138. Valid input keywords and parameter values

Keyword	Valid values	Default value	Explanation
mem_percent	1–100	25	
workload_type	simple, mixed, complex	mixed	Simple workloads tend to be I/O intensive and mostly transactions, whereas complex workloads tend to be CPU intensive and mostly queries.
num_stmts	1–1 000 000	25	Number of statements per unit of work
tpm	1–200 000	60	Transactions per minute
admin_priority	performance, recovery, both	both	Optimize for better performance (more transactions per minute) or better recovery time
num_local_apps	0–5 000	0	Number of connected local applications
num_remote_apps	0–5 000	100	Number of connected remote applications
isolation	RR, RS, CS, UR	RR	Isolation level of applications connecting to this database (Repeatable Read, Read Stability, Cursor Stability, Uncommitted Read)
bp_resizeable	yes, no	yes	Are buffer pools resizeable?

APPLY

DB ONLY

Displays the recommended values for the database configuration and the buffer pool settings based on the current database manager configuration. Applies the recommended changes to the database configuration and the buffer pool settings.

DB AND DBM
Displays and applies the recommended changes to the database manager configuration, the database configuration, and the buffer pool settings.

NONE
Disables the Configuration Advisor (it is enabled by default).

- If the AUTOCONFIGURE keyword is specified with the CREATE DATABASE command, the DB2_ENABLE_AUTOCONFIG_DEFAULT variable value is not considered. Adaptive Self Tuning Memory and Auto Runstats will be enabled and the Configuration Advisor will tune the database configuration and database manager configuration parameters as indicated by the APPLY DB or APPLY DBM options.
- Specifying the AUTOCONFIGURE option with the CREATE DATABASE command on a database will recommend enablement of the Self Tuning Memory Manager. However, if you run the AUTOCONFIGURE command on a database in an instance where SHEAPTHRES is not zero, sort memory tuning (SORTHEAP) will not be enabled automatically. To enable sort memory tuning (SORTHEAP), you must set SHEAPTHRES equal to zero using the UPDATE DATABASE MANAGER CONFIGURATION command. Note that changing the value of SHEAPTHRES may affect the sort memory usage in your previously existing databases.

Examples

Here are several examples of the CREATE DATABASE command:

Example 1:

```
CREATE DATABASE TESTDB3
 AUTOMATIC STORAGE YES
```

Database TESTDB3 is created on the drive that is the value of database manager configuration parameter *dftdbpath*. Automatic storage is enabled with a single storage path that also has the value of *dftdbpath*.

Example 2:

```
CREATE DATABASE TESTDB7 ON C:,D:
```

Database TESTDB7 is created on drive C: (first drive in storage path list). Automatic storage is implicitly enabled and the storage paths are C: and D:.

Example 3:

```
CREATE DATABASE TESTDB15
 AUTOMATIC STORAGE YES
 ON C:,D: DBPATH ON E:
```

Database TESTDB15 is created on drive E: (explicitly listed as DBPATH). Automatic storage is explicitly enabled and the storage paths are C: and D:.

Usage notes

The CREATE DATABASE command:

- Creates a database in the specified subdirectory. In a partitioned database environment, creates the database on all database partitions listed in db2nodes.cfg, and creates a $DB2INSTANCE/NODE*xxxx* directory under the specified

subdirectory at each database partition. In a single partition database environment, creates a `$DB2INSTANCE/NODE0000` directory under the specified subdirectory.

- Creates the system catalog tables and recovery log.
- Catalogs the database in the following database directories:
  - Server's local database directory on the path indicated by *path* or, if the path is not specified, the default database path defined in the database manager system configuration file by the *dftdbpath* parameter. A local database directory resides on each file system that contains a database.
  - Server's system database directory for the attached instance. The resulting directory entry will contain the database name and a database alias.

    If the command was issued from a remote client, the client's system database directory is also updated with the database name and an alias.

  Creates a system or a local database directory if neither exists. If specified, the comment and code set values are placed in both directories.

  Note: If the change the database by path configuration parameter **newlogpath** is not set, the default for the location of log files configuration parameter **logpath** is the path shown by the DBPATH ON option. It is suggested that the DBPATH ON option be used when automatic storage is enabled to keep the database information separate from the database data.
- Stores the specified code set, territory, and collating sequence. A flag is set in the database configuration file if the collating sequence consists of unique weights, or if it is the identity sequence.
- Creates the schemas called SYSCAT, SYSFUN, SYSIBM, and SYSSTAT with SYSIBM as the owner. The database partition server on which this command is issued becomes the catalog database partition for the new database. Two database partition groups are created automatically: IBMDEFAULTGROUP and IBMCATGROUP.
- Binds the previously defined database manager bind files to the database (these are listed in the utilities bind file list, `db2ubind.lst`). If one or more of these files do not bind successfully, CREATE DATABASE returns a warning in the SQLCA, and provides information about the binds that failed. If a bind fails, the user can take corrective action and manually bind the failing file. The database is created in any case. A schema called NULLID is implicitly created when performing the binds with CREATEIN privilege granted to PUBLIC, if the RESTRICTIVE option is not selected.

  The utilities bind file list contains two bind files that cannot be bound against previous version of the server:
  - `db2ugtpi.bnd` cannot be bound against DB2 Version 2 servers.
  - `db2dropv.bnd` cannot be bound against DB2 Parallel Edition Version 1 servers.

  If `db2ubind.lst` is bound against a server which is not at the latest level, warnings pertaining to these two files are returned, and can be disregarded.
- Creates SYSCATSPACE, TEMPSPACE1, and USERSPACE1 table spaces. The SYSCATSPACE table space is only created on the catalog database partition.
- Grants the following:
  - EXECUTE WITH GRANT privilege to PUBLIC on all functions in the SYSFUN schema
  - EXECUTE privilege to PUBLIC on all procedures in SYSIBM schema
  - DBADM, CONNECT, CREATETAB, BINDADD, CREATE_NOT_FENCED, IMPLICIT_SCHEMA and LOAD authorities to the database creator

- CONNECT, CREATETAB, BINDADD, and IMPLICIT_SCHEMA authorities to PUBLIC
- USE privilege on the USERSPACE1 table space to PUBLIC
- SELECT privilege on each system catalog to PUBLIC
- BIND and EXECUTE privilege to PUBLIC for each successfully bound utility.
- EXECUTE WITH GRANT privilege to PUBLIC on all functions in the SYSFUN schema.
- EXECUTE privilege to PUBLIC on all procedures in SYSIBM schema.

Note: If the RESTRICTIVE option is present it causes the RESTRICT_ACCESS database configuration parameter to be set to YES and no privileges or authorities are automatically granted to PUBLIC. For additional information, see the RESTRICTIVE option of the CREATE DATABASE command.

Automatic storage is a collection of storage paths associated with a database on which table spaces can be created without having to explicitly specify container definitions (see the *CREATE TABLESPACE statement* for more information). Automatic storage is enabled by default, but can be explicitly disabled for a database when it is created. Automatic storage can be disabled at database creation time by specifying the AUTOMATIC STORAGE NO option.

It is important to note that automatic storage can only be enabled at database creation time, it cannot be enabled after the database has been created. Also, automatic storage cannot be disabled once a database has been defined to use it.

When free space is calculated for an automatic storage path for a given database partition, the database manager will check for the existence of the following directories or mount points within the storage path and will use the first one that is found. In doing this, file systems can be mounted at a point beneath the storage path and the database manager will recognize that the actual amount of free space available for table space containers may not be the same amount that is associated with the storage path directory itself.

1. `<storage path>/<instance name>/NODE####/<database name>`
2. `<storage path>/<instance name>/NODE####`
3. `<storage path>/<instance name>`
4. `<storage path>/<`

Where

- `<storage path>` is a storage path associated with the database.
- `<instance name>` is the instance under which the database resides.
- `NODE####` corresponds to the database partition number (for example NODE0000 or NODE0001).
- `<database name>` is the name of the database.

Consider the example where two logical database partitions exist on one physical machine and the database is being created with a single storage path: `/db2data`. Each database partition will use this storage path but the user may want to isolate the data from each partition within its own file system. In this case, a separate file system can be created for each partition and be mounted at `/db2data/<instance>/NODE####`. When creating containers on the storage path and determining free space, the database manager will know not to retrieve free space information for `/db2data`, but instead retrieve it for the corresponding `/db2data/<instance>/NODE####` directory.

In general, the same storage paths must be used for each partition in a multi-partition database and they must all exist prior to executing the CREATE DATABASE command. One exception to this is where database partition expressions are used within the storage path. Doing this allows the database partition number to be reflected in the storage path such that the resulting path name is different on each partition.

You use the argument `" $N" ([blank]$N)` to indicate a database partition expression. A database partition expression can be used anywhere in the storage path, and multiple database partition expressions can be specified. Terminate the database partition expression with a space character; whatever follows the space is appended to the storage path after the database partition expression is evaluated. If there is no space character in the storage path after the database partition expression, it is assumed that the rest of the string is part of the expression. The argument can only be used in one of the following forms:

Operators are evaluated from left to right. `%` represents the modulus operator. The database partition number in the examples is assumed to be 10.

Syntax	Example	Value
[blank]$N	″ $N″	10
[blank]$N+[number]	″ $N+100″	110
[blank]$N%[number]	″ $N%5″	0
[blank]$N+[number]%[number]	″ $N+1%5″	1
[blank]$N%[number]+[number]	″ $N%4+2″	4
[a] % is modulus.		

With *dbadm* authority, one can grant these privileges to (and revoke them from) other users or PUBLIC. If another administrator with *sysadm* or *dbadm* authority over the database revokes these privileges, the database creator nevertheless retains them.

In an MPP environment, the database manager creates a subdirectory, `$DB2INSTANCE/NODE`*xxxx*, under the specified or default path on all database partitions. The *xxxx* is the database partition number as defined in the `db2nodes.cfg` file (that is, database partition 0 becomes `NODE0000`). Subdirectories `SQL00001` through `SQL`*nnnnn* will reside on this path. This ensures that the database objects associated with different database partitions are stored in different directories (even if the subdirectory `$DB2INSTANCE` under the specified or default path is shared by all database partitions).

If LDAP (Lightweight Directory Access Protocol) support is enabled on the current machine, the database will be automatically registered in the LDAP directory. If a database object of the same name already exists in the LDAP directory, the database is still created on the local machine, but a warning message is returned, indicating that there is a naming conflict. In this case, the user can manually catalog an LDAP database entry by using the CATALOG LDAP DATABASE command.

CREATE DATABASE will fail if the application is already connected to a database.

When a database is created, a detailed deadlocks event monitor is created. As with any monitor, there is some overhead associated with this event monitor. You can drop the deadlocks event monitor by issuing the DROP EVENT MONITOR command.

Use CATALOG DATABASE to define different alias names for the new database.

The combination of the code set and territory values must be valid. For a list of the supported combinations, see *Supported territory codes and code pages*.

To specify a database path (instead of a drive) on a Windows system, you need to set the DB2 registry variable: `DB2_CREATE_DB_ON_PATH=YES`.

Compatibilities

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.

db2audit - Audit facility administrator tool

DB2 database systems provide an audit facility to assist in the detection of unknown or unanticipated access to data. The DB2 audit facility generates and permits the maintenance of an audit trail for a series of predefined database events.

The records generated from this facility are kept in audit log files. The analysis of these records can reveal usage patterns which would identify system misuse. Once identified, actions can be taken to reduce or eliminate such system misuse. The audit facility acts at both the instance and database levels, independently recording all activities in separate logs based on either the instance or the database.

DB2 database systems provide the ability to independently audit at both the instance and at the individual database level. The db2audit tool is used to configure audit at the instance level as well as control when such audit information is collected. The AUDIT SQL statement is used to configure and control the audit requirements for an individual database. The db2audit tool can be used to archive both instance and database audit logs as well as to extract from archived logs of either type.

When working in a partitioned database environment, many of the auditable events occur at the database partition at which the user is connected (the coordinator partition) or at the catalog partition (if they are not the same database partition). The implication of this is that audit records can be generated by more than one database partition. Part of each audit record contains information on the coordinator partition and originating database partition identifiers.

The instance audit log (`db2audit.instance.log.`*`node_number`*`[.`*`timestamp`*`]`) is located in the instance's `security/auditdata` subdirectory, and the audit configuration file (`db2audit.cfg`) is located in the instance's `security` subdirectory. The database audit log is named `db2audit.db.`*`dbname`*`.log.`*`node_number`*`[.`*`timestamp`*`]`. At the time you create an instance, read/write permissions are set on these files, where possible, by the operating system. By default, the permissions are read/write for the instance owner only. It is recommended that you do not change these permissions.

Authorized users of the audit facility can control the following actions within the audit facility, using db2audit:

- Start recording auditable events within the DB2 instance. This does not include database level activities.
- Stop recording auditable events within the DB2 instance.
- Configure the behavior of the audit facility at the instance level only.
- Select the categories of the auditable events to be recorded at the instance level only.
- Request a description of the current audit configuration for the instance.
- Flush any pending audit records from the instance and write them to the audit log.
- Archive audit records from the current audit log for either the instance or a database under the instance.
- Extract audit records from an archived audit log by formatting and copying them to a flat file or ASCII delimited file. Extraction is done in preparation for analysis of log records.

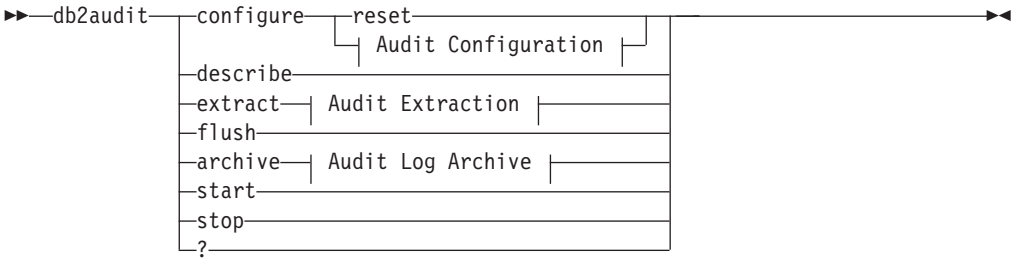
Authorization

SYSADM

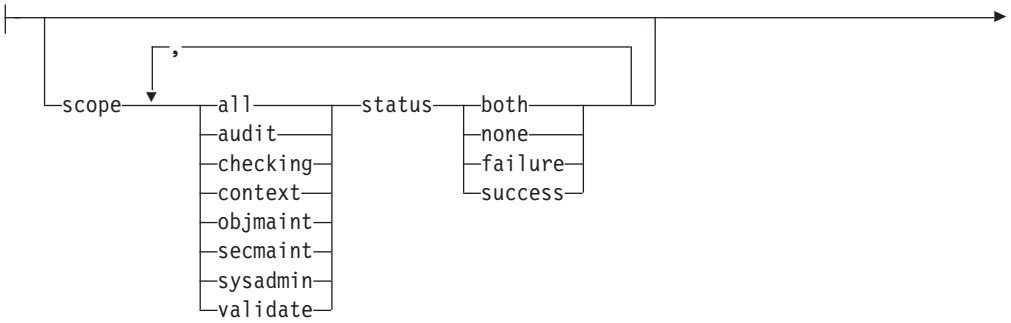
Required Connection

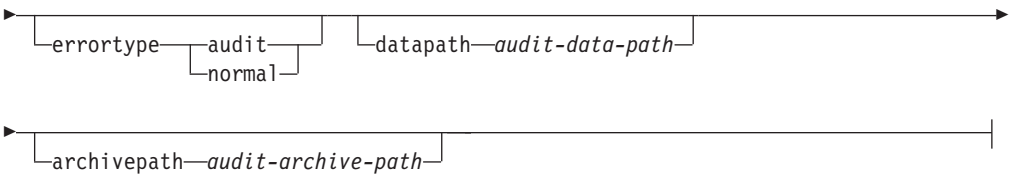
None

Command syntax

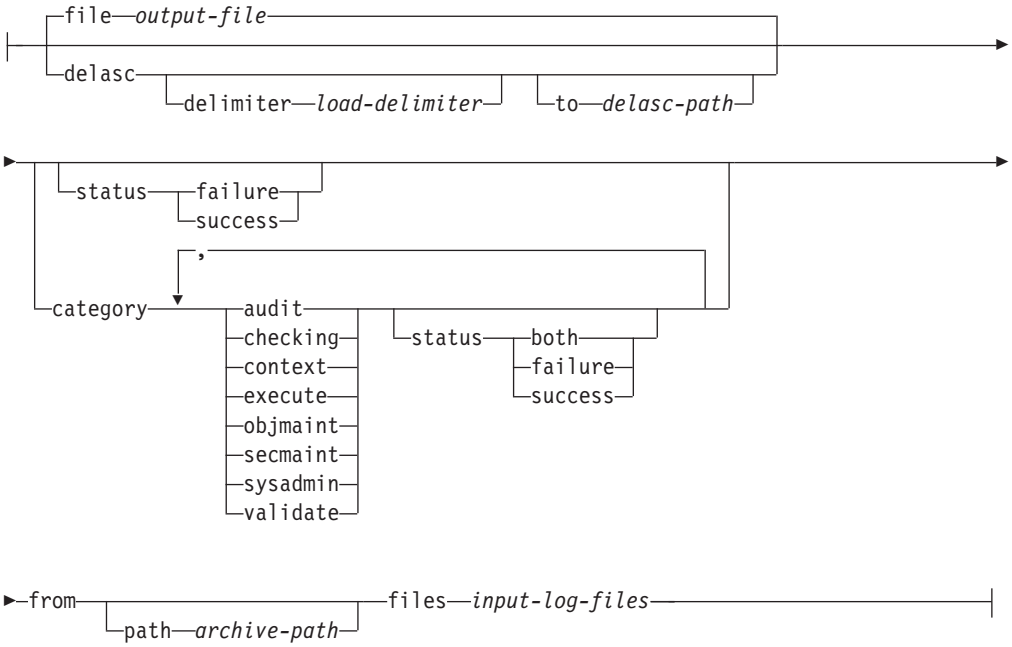


Audit Configuration:

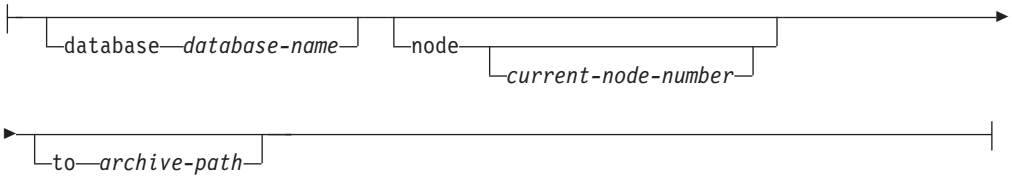




Audit Extraction:



Audit Log Archive:



Command parameters

configure

This parameter allows the modification of the `db2audit.cfg` configuration file in the instance's `security` subdirectory. Updates to this file can occur even when the instance is stopped. Updates, occurring when the instance is active, dynamically affect the auditing being done by the DB2 instance. The configure action on the configuration file causes the creation of an audit record if the audit facility has been started and the **audit** category of auditable events is being audited. All configure options, except the data path and archive path, only apply to instance level audit events, and not to database level audit events. The path options apply to the instance and all databases within the instance.

The following are the possible actions on the configuration file:

reset This action causes the configuration file to revert to the initial configuration (where **scope** is all of the categories except context, **status** for each category is failure, **errortype** is normal, and the auditing of instance level events is off). This action will create a new audit configuration file if the original has been lost or damaged. The audit data path and archive path will be blank. This option does not reset any of the audit policies or use of those policies at the database level.

scope This action specifies which categories will be audited, and the status of each of those categories.

status This action specifies whether only successful or failing events, or both successful and failing events, should be logged. **status** has the following options:

both Successful and failing events will be audited.

none No events for this category will be audited.

failure
Only failing events will be audited.

success
Only successful events will be audited.

Only the categories specified on the configure statement will be modified. All other categories will have their status preserved.

Note:

- The default **scope** is all categories except **context** and may result in records being generated rapidly. In conjunction with the mode (synchronous or asynchronous), the selection of the categories may result in a significant performance reduction and significantly increased disk requirements. It is recommended that the number and type of events being logged be limited as much as possible, otherwise the size of the audit log will grow rapidly. This action also allows a particular focus for auditing and reduces the growth of the log.
- **context** events occur before the status of an operation is known. Therefore, such events are logged regardless of the value associated with this parameter, unless the **status** is none.
- If the same category is repeated, or categories are also specified with the all keyword, a syntax error will be returned.

errortype
This action specifies whether audit errors are returned to the user or are ignored. The value for this parameter can be:

audit All errors including errors occurring within the audit facility are managed by DB2 database and all negative SQLCODEs are reported back to the caller.

normal
Any errors generated by db2audit are ignored and only the SQLCODEs for the errors associated with the operation being performed are returned to the application.

datapath *audit-data-path*
This is the directory to which the audit logs produced by the DB2 database system will be written. The default is

sqllib/security/auditdata (*instance path**instance*\security\auditdata on Windows). This parameter affects all auditing within an instance, including database level auditing. This must be a fully qualified path and not a relative path. The instance owner must have write permission on this directory. On Windows, the user issuing a local instance command, for example, db2start, db2audit, and db2 update dbm cfg, must also have write permission on this directory if the command is required to be audited. On a partitioned database environment, this directory does not need to be an NFS shared directory, although that is possible. A non-shared directory will result in increased performance as each node is writing to a unique disk. The maximum length of the path is 971 bytes for UNIX or Linux and 208 bytes for Windows operating systems.

If the path is provided as "", then the path will be updated to be the default. db2audit describe will show no path as being set and the default path will be used. Note, to prevent the shell from interpreting the quotes, they will generally need to be escaped, for example

```
db2audit configure datapath \"\"
```

The data path must exist. In a partitioned database environment, the same data path will be used on each node. There is no way to specify a unique set of data paths for a particular node unless database partition expressions are used as part of the data path name. Doing this allows the node number to be reflected in the storage path such that the resulting path name is different on each database partition.

archivepath *audit-archive-path*

This is the default directory for the archive and extract options. In a partitioned database environment, it is recommended that this directory be an NFS shared directory accessible by all nodes. The default is sqllib/security/auditdata (sqllib*instance*\security\auditdata on Windows). This must be a fully qualified path and not a relative path. The instance owner must have write permission on this directory. The maximum length of the path is 971 bytes for UNIX or Linux and 208 bytes for Windows operating systems.

The archive path must exist, and database partition expressions are NOT allowed for the archive path.

describe

This parameter displays to standard output the current instance level audit configuration information and status.

The following items are displayed:

- If audit is active.
- The status for each category.
- The error type in the form of whether or not an SQLCA is returned on errors.
- The data and archive paths.

This is an example of what the **describe** output looks like:

```
DB2 AUDIT SETTINGS:

Audit active: "FALSE "
Log audit events: "SUCCESS"
```

```
Log checking events: "FAILURE"
Log object maintenance events: "BOTH"
Log security maintenance events: "BOTH "
Log system administrator events: "NONE"
Log validate events: "FAILURE"
Log context events: "NONE"
Return SQLCA on audit error: "TRUE "
Audit Data Path: "/auditdata"
Audit Archive Path: "/auditarchive"

AUD0000I  Operation succeeded.
```

extract This parameter allows the movement of audit records from the audit log to an indicated destination. The audit log will be created in the database code page. All of the fields will be converted to the current application code page when extract is run.

The following are the options that can be used when extracting:

file *output-file*
The extracted audit records are placed in *output-file*. If the directory is not specified, *output-file* is written to the current working directory. If the file already exists the output will be appended to it. If a file name is not specified, records are written to the `db2audit.out` file in the archive path specified in the audit configuration file.

delasc
The extracted audit records are placed in a delimited ASCII format suitable for loading into DB2 database relational tables. The output is placed in separate files, one for each category. In addition, the file `auditlobs` will also be created to hold any lobs that are included in the audit data. The filenames are:

- `audit.del`
- `checking.del`
- `objmaint.del`
- `secmaint.del`
- `sysadmin.del`
- `validate.del`
- `context.del`
- `execute.del`
- `auditlobs`

If the files already exist the output will be appended to them. The `auditlobs` file will be created if the **context** or **execute** categories are extracted. LOB Location Specifiers are included in the `.del` files to reference the LOBS in the `auditlobs` file.

delimiter *load-delimiter*
Allows you to override the default audit character string delimiter, which is the double quote ("), when extracting from the audit log. You would use **delimiter** followed by the new delimiter that you want to use in preparation for loading into a table that will hold the audit records. The new load delimiter can be either a single character (such as !) or a four-character string representing a hexadecimal number (such as `0xff`).

to *delasc-path*
Allows you to specify the path to which the delimited files are

written. If it is not specified, then the files are written to the directory indicated by the audit archive path option specified in the audit configuration file.

category
The audit records for the specified categories of audit events are to be extracted. If not specified, all categories are eligible for extraction.

status
The audit records for the specified status are to be extracted. If not specified, all records are eligible for extraction.

path
The path to the location of the archived audit logs. If this is not specified, the archive path in the audit configuration will be used. The path is not used if the filename contains a fully qualified path.

files
The list of audit log files that will be extracted. This may be a single file or a list of files. These files are not altered during an extract. The filenames will be combined with **path** to get the fully qualified filenames if they are not already fully qualified. The list may included standard shell wild cards to specify multiple files.

flush This parameter forces any pending audit records to be written to the audit log. Also, the audit state is reset from ″unable to log″ to a state of ″ready to log″ if the audit facility is in an error state.

archive
This parameter moves the current audit log for either an individual database or the instance to a new location for archiving and later extraction. The current timestamp will be appended to the filename. All records that are currently being written to the audit log will complete before the log is archived to ensure full records are not split apart. All records that are created while the archive is in progress will be written to the current audit log, and not the archived log, once the archive has finished.

The following are the options that can be used when archiving:

database *database-name*
The name of the database for which you would like to archive the audit log. If the database name is not supplied, then the instance level audit log is archived.

node
Indicates that the archive command is to only be run on the current node, and that the **node_number** monitor element will indicate what the current node is. This is only required on a partitioned database environment.

current-node-number
Informs the db2audit executable about which node it is currently running on. This parameter is required if the **DB2NODE** environment variable does not contain the current node.

to *archive-path*
The directory where the archived audit log should be created. The directory must exist and the instance owner must have create permission on this directory. If this is not provided, the archive path in the audit configuration will be used.

The format of the filename that is created is:

- `db2audit.instance.log.`*`node_number`*`[.`*`YYYYMMDDHHMMSS`*`]` for the instance log
- `db2audit.db.`*`dbname`*`.log.`*`node_number`*`[.`*`YYYYMMDDHHMMSS`*`]` for the database log

where *YYYY* is the year, *MM* is the month, *DD* is the day, *HH* is the hour, *MM* is the minute, and *SS* is the seconds. The time will be the local time. The database name portion will not be present for instance audit logs. The node number in a non-partitioned database environment will be 0. If the file already exists, an append will be performed.

The timestamp will not reflect the last record in the log with 100% accuracy. The timestamp represents when the archive command was run. Entries that are currently being written to the log file must finish before it can be moved, and these entries may have timestamps that are later than the timestamp given to the filename.

If the **node** option is not specified, then the audit log on all nodes will be archived. The database server must be started in this case. If the database server has not been started, then archive must be run on each node, and the **node** option must be specified to indicate on which node **archive** is to be run (AUD0029).

The **archive** option will output the result and names of the files from each node that archive was run on.

start This parameter causes the audit facility to begin auditing events based on the contents of the `db2audit.cfg` file for the instance only. In a partitioned DB2 database instance, auditing will begin for instance and client level activities on all database partitions when this clause is specified. If the **audit** category of events has been specified for auditing, then an audit record will be logged when the audit facility is started. This has no effect on database level auditing, which is controlled through the AUDIT DDL statement.

stop This parameter causes the audit facility to stop auditing events for the instance only. In a partitioned DB2 database instance, auditing will be stopped for instance and client level activities on all database partitions when this clause is specified. If the **audit** category of events has been specified for auditing, then an audit record will be logged when the audit facility is stopped. This has no effect on database level auditing, which is controlled through the AUDIT DDL statement.

? This parameter displays the help information for the db2audit command.

Examples

This is a typical example of how to archive and extract a delimited ASCII file in a partitioned database environment. The Windows remove (rm) command deletes the old delimited ASCII files.

```
rm /auditdelasc/*.del
db2audit flush
db2audit archive database mydb to /auditarchive
```

(files will be indicated for use in next step)

```
db2audit extract delasc to /auditdelasc from files /auditarchive
/db2audit.db.mydb.log.*.20070514102856
```

Load the .del files into a DB2 table.

Usage notes

- Database level auditing is controlled with the AUDIT statement.
- The instance level audit facility must be stopped and started explicitly. When starting, the audit facility uses existing audit configuration information. Since the audit facility is independent of the DB2 database server, it will remain active even if the instance is stopped. In fact, when the instance is stopped, an audit record may be generated in the audit log.
- Ensure that the audit facility has been turned on by issuing the db2audit start command before using the audit utilities.
- There are different categories of audit records that may be generated. In the description of the categories of events available for auditing (below), you should notice that following the name of each category is a one-word keyword used to identify the category type. The categories of events available for auditing are:
  - Audit (**audit**). Generates records when audit settings are changed or when the audit log is accessed.
  - Authorization Checking (**checking**). Generates records during authorization checking of attempts to access or manipulate DB2 database objects or functions.
  - Object Maintenance (**objmaint**). Generates records when creating or dropping data objects.
  - Security Maintenance (**secmaint**). Generates records when granting or revoking: object or database privileges, or DBADM authority. Records are also generated when the database manager security configuration parameters **sysadm_group**,**sysctrl_group**, or **sysmaint_group** are modified.
  - System Administration (**sysadmin**). Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.
  - User Validation (**validate**). Generates records when authenticating users or retrieving system security information.
  - Operation Context (**context**). Generates records to show the operation context when an instance operation is performed. This category allows for better interpretation of the audit log file. When used with the log's event correlator field, a group of events can be associated back to a single database operation.
  - You can audit failures, successes, both or none.
- Any operation on the instance may generate several records. The actual number of records generated and moved to the audit log depends on the number of categories of events to be recorded as specified by the audit facility configuration. It also depends on whether successes, failures, or both, are audited. For this reason, it is important to be selective of the events to audit.
- To clean up and/or view audit logs, run **archive** on a regular basis, then run **extract** on the archived file to save what is useful. The audit logs can then be deleted with standard file system delete commands.

db2gpmap - Get distribution map

If a database is already set up and database partition groups defined for it, db2gpmap gets the distribution map for the database table or the database partition group from the catalog partitioned database server.

Authorization

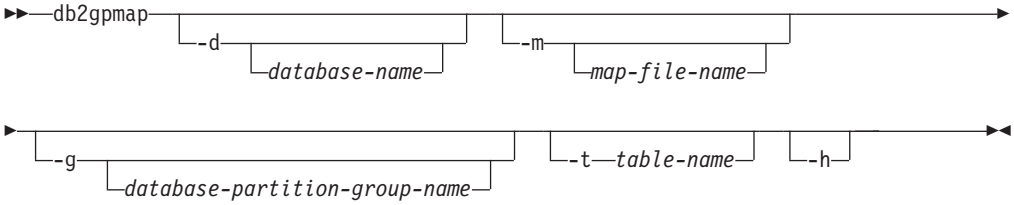
Both of the following:

- Read access to the system catalog tables
- BIND and EXECUTE package privileges on `db2gpmap.bnd`

Required connection

Before using db2gpmap the database manager must be started and `db2gpmap.bnd` must be bound to the database. If not already bound db2gpmap will attempt to bind the file.

Command syntax



Command parameters

-d *database-name*
Specifies the name of the database for which to generate a distribution map. If no database name is specified, the value of the DB2DBDFT environment variable is used. If DB2DBDFT is not set, the default is the SAMPLE database.

-m *map-file-name*
Specifies the fully qualified file name where the distribution map will be saved. The default is `db2split.map`.

-g *database-partition-group-name*
Specifies the name of the database partition group for which to generate a distribution map. The default is IBMDEFAULTGROUP.

-t *table-name*
Specifies the table name.

-h Displays usage information.

Examples

The following example extracts the distribution map for a table ZURBIE.SALES in database SAMPLE into a file called `C:\pmaps\zurbie_sales.map`:

```
db2gpmap -d SAMPLE -m C:\pmaps\zurbie_sales.map -t ZURBIE.SALES
```

db2icrt - Create instance

Creates DB2 instances.

On Linux and UNIX operating systems, this utility is located in the *DB2DIR*`/instance` directory, where *DB2DIR* represents the installation location where the current version of the DB2 database system is installed. On Windows operating systems, this utility is located under the **DB2PATH**`\bin` directory where **DB2PATH** is the location where the DB2 copy is installed.

The db2icrt command creates DB2 instances in the instance owner's home directory.

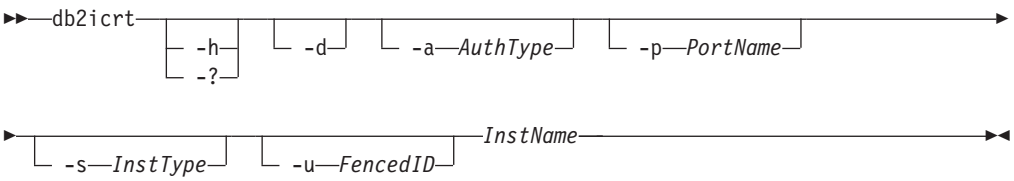
Note: This command is not available for a non-root installation of DB2 database products on Linux and UNIX operating systems.

Authorization

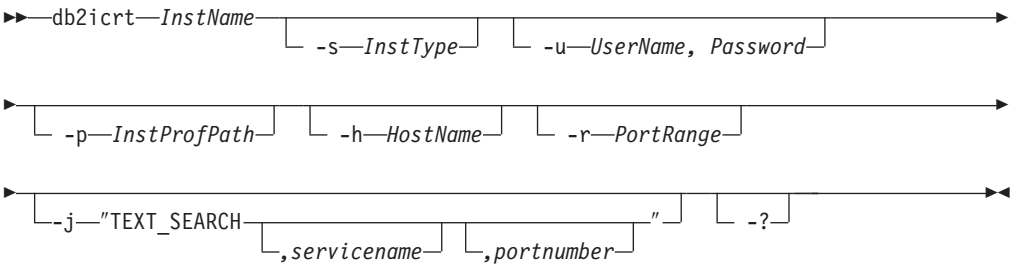
Root access on Linux and UNIX operating systems or Local Administrator authority on Windows operating systems.

Command syntax

For Linux and UNIX operating systems



For Windows operating systems



Command parameters

For Linux and UNIX operating systems

-h | -? Displays the usage information.

-d Turns debug mode on. Use this option only when instructed by DB2 Support.

-a *AuthType*
Specifies the authentication type (SERVER, CLIENT or SERVER_ENCRYPT) for the instance. The default is SERVER.

-p *PortName*
Specifies the port name or number used by the instance. This option does not apply to client instances.

-s *InstType*
Specifies the type of instance to create. Use the **-s** option only when you are creating an instance other than the default associated with the installed product from which you are running db2icrt. Valid values are:

client Used to create an instance for a client. This is the default instance

type for IBM Data Server Client, IBM Data Server Runtime Client, and DB2 Connect Personal Edition.

standalone
Used to create an instance for a database server with local clients. It is the default instance type for DB2 Personal Edition.

ese Used to create an instance for a database server with local and remote clients with DPF support. This is the default instance type for DB2 Enterprise Server Edition.

wse Used to create an instance for a database server with local and remote clients. This is the default instance type for DB2 Workgroup Server Edition, DB2 Express or Express-C Edition, and DB2 Connect Enterprise Edition.

DB2 products support their default instance types and the instance types lower than their default ones. For instance, DB2 Enterprise Server Edition supports the instance types of ese, wse, standalone and client.

-u *Fenced ID*
Specifies the name of the user ID under which fenced user-defined functions and fenced stored procedures will run. The **-u** option is required if you are not creating a client instance.

InstName
Specifies the name of the instance which is also the name of an existing user in the operating system. This has to be the last argument of the db2icrt command.

For Windows operating systems

InstName
Specifies the name of the instance.

-s *InstType*
Specifies the type of instance to create. Currently, there are four kinds of DB2 instance types. Valid values are:

client Used to create an instance for a client. This is the default instance type for IBM Data Server Client, IBM Data Server Runtime Client, and DB2 Connect Personal Edition.

standalone
Used to create an instance for a database server with local clients. It is the default instance type for DB2 Personal Edition.

ese Used to create an instance for a database server with local and remote clients with DPF support. The

```
-s ese -u Username, Password
```

options have to be used with db2icrt to create the ESE instance type and a DPF instance.

wse Used to create an instance for a database server with local and remote clients. This is the default instance type for DB2 Workgroup Server Edition, DB2 Express or Express-C Edition, and DB2 Connect Enterprise Edition.

DB2 products support their default instance types and the instance types lower than their default ones. For instance, DB2 Enterprise Server Edition supports the instance types of ese, wse, standalone and client.

-u *Username, Password*
Specifies the account name and password for the DB2 service. This option is required when creating a partitioned database instance.

-p *InstProfPath*
Specifies the instance profile path.

-h *HostName*
Overrides the default TCP/IP host name if there is more than one for the current machine. The TCP/IP host name is used when creating the default database partition (database partition 0). This option is only valid for partitioned database instances.

-r *PortRange*
Specifies a range of TCP/IP ports to be used by the partitioned database instance when running in MPP mode. For example, `-r 50000,50007`. The services file of the local machine will be updated with the following entries if this option is specified:

```
DB2_InstName          baseport/tcp
DB2_InstName_END      endport/tcp
```

/j "**TEXT_SEARCH**"
Configures the DB2 Text Search server using generated default values for service name and TCP/IP port number. This parameter cannot be used if the instance type is *client*.

/j "**TEXT_SEARCH**, *servicename*"
Configures the DB2 Text Search server using the provided service name and an automatically generated port number. If the service name has a port number assigned in the services file, it uses the assigned port number.

/j "**TEXT_SEARCH**, *servicename, portnumber*"
Configures the DB2 Text Search server using the provided service name and port number.

/j "**TEXT_SEARCH**, *portnumber*"
Configures the DB2 Text Search server using a default service name and the provided port number. Valid port numbers must be within the 1024 - 65535 range.

-? Displays usage information.

Examples

- On an AIX machine, to create an instance for the user ID `db2inst1`, issue the following command:

  On a client machine:

  ```
  DB2DIR/instance/db2icrt db2inst1
  ```

  On a server machine:

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

  where `db2fenc1` is the user ID under which fenced user-defined functions and fenced stored procedures will run.

Usage notes

- The `instance home/sqllib/db2tss/config` folder is created by db2icrt on Linux and UNIX operating systems. It is advised that you use a symbolic link to an area outside the `sqllib` directory.
- The **-s** option is intended for situations in which you want to create an instance that does not use the full functionality of the system. For example, if you are using Enterprise Server Edition (ESE) on a UNIX operating system, but do not want partition capabilities, you could create a Workgroup Server Edition (WSE) instance, using the option `-s WSE`.
- To create a DB2 instance that supports Microsoft Cluster Server, first create an instance, then use the db2mscs command to migrate it to run in a MSCS instance.
- On Linux and UNIX operating systems, only one instance can be created under a user name. If you want to create an instance under a user name that already has an instance, you must drop the existing instance before creating the new one.
- When creating DB2 instances, consider the following restrictions:
  - If existing IDs are used to create DB2 instances, make sure that the IDs are not locked and do not have passwords expired.
- You can also use the db2isetup command to create and update DB2 instances using a graphical interface on all supported Linux and UNIX operating systems.
- On Linux and UNIX systems, if you are using the su command instead of the login command to become the root user, you must issue the su command with the **-** option to indicate that the process environment is to be set as if you had logged in to the system using the login command.
- On Linux and UNIX operating systems, you must not source the DB2 instance environment for the root user. Running db2icrt when you sourced the DB2 instance environment is not supported.
- On AIX 6.1 (or higher), when running this command from a shared DB2 copy in a system workload partition (WPAR) global environment, this command must be run as the root user.
- On Windows operating systems, if the IBM Tivoli Monitoring for Databases: DB2 Agent is installed and the DB2 instance is created, the Monitoring Agent for DB2 instance is also created if the following are true:
  - The DB2 instance type is standalone, wse, or ese.
  - The default DB2 copy has the ITM agent component installed.
  - The DB2 instance is at version 9.5 (or higher).
  - There is no existing ITM for Databases product.

  In addition, the following are also created after the creation of the Monitoring Agent for DB2 instance: the Monitoring Agent for DB2 instance files, the NT service and the registry entries.

db2iupdt - Update instances

This command updates an instance to run on a DB2 copy that has a new DB2 database product or feature installed, to run on a DB2 copy of the same version as the DB2 copy associated with the instance, or to upgrade the instance type to a higher level instance type.

The db2iupdt command can be issued against instances of the same version that are associated with the same or a different DB2 copy. In all cases, it will update the

instance so that it runs against the code located in the same DB2 copy as where you issued the db2iupdt command. You should issue this command:

- if you install a fix pack and the automatic instance update fails.
- if you install a new DB2 database product or feature to the DB2 copy associated to the DB2 instance.
- if you want to update a DB2 instance from one DB2 copy to another DB2 copy of the same version of DB2 database product.

After a fix pack is installed on Linux and UNIX operating systems, the db2iupdt command is executed automatically.

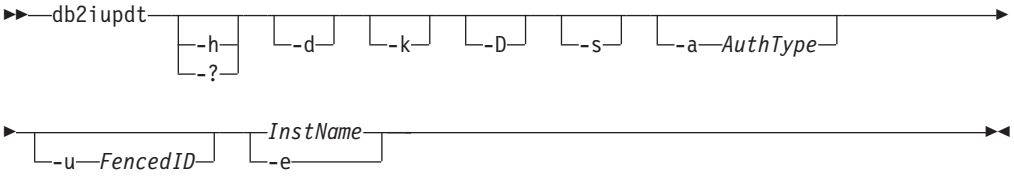
To update an instance with db2iupdt, you must first stop all processes that are running for the instance.

Authorization

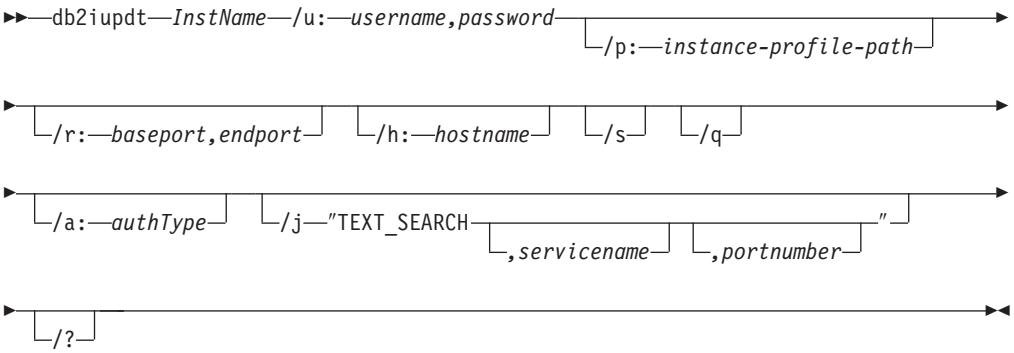
`Root` access on UNIX and Linux operating systems or Local Administrator on Windows operating systems.

Command syntax

For UNIX and Linux operating systems



For Windows operating systems



Command parameters

For UNIX and Linux operating systems

-h | -? Displays the usage information.

-d Turns debug mode on.

-k Keeps the current instance type during the update.

-D Moves an instance from a higher code level on one path to a lower code level installed on another path.

-s Ignores the existing SPM log directory.

-a *AuthType*
Specifies the authentication type (SERVER, SERVER_ENCRYPT or CLIENT) for the instance. The default is SERVER.

-u *Fenced ID*
Specifies the name of the user ID under which fenced user defined functions and fenced stored procedures will run. This option is only needed when converting an instance from a client instance to a non-client instance type. To determine the current instance type, refer to the node type parameter in the output from a GET DBM CFG command. If an instance is already a non-client instance, or if an instance is a client instance and is staying as a client instance (for example, by using the **-k** option), the **-u** option is not needed. The **-u** option cannot change the fenced user for an existing instance.

InstName
Specifies the name of the instance.

-e Updates every instance.

For Windows operating systems

InstName
Specifies the name of the instance.

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

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

/r:*baseport,endport*
Specifies the range of TCP/IP ports to be used by the partitioned database instance when running in MPP mode. When this option is specified, the services file on the local machine will be updated with the following entries:

```
DB2_InstName       baseport/tcp
DB2_InstName_END   endport/tcp
```

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

/s Updates the instance to a partitioned instance.

/q Issues the db2iupdt command in quiet mode.

/a:*authType*
Specifies *authType*, the authentication type (SERVER, CLIENT, or SERVER_ENCRYPT) for the instance.

/j "**TEXT_SEARCH**"
Configures the DB2 Text Search server using generated default values for service name and TCP/IP port number. This parameter cannot be used if the instance type is *client*.

/j "**TEXT_SEARCH**, *servicename*"

Configures the DB2 Text Search server using the provided service name and an automatically generated port number. If the service name has a port number assigned in the services file, it uses the assigned port number.

/j *"***TEXT_SEARCH***, servicename, portnumber"*

Configures the DB2 Text Search server using the provided service name and port number.

/j *"***TEXT_SEARCH***, portnumber"*

Configures the DB2 Text Search server using a default service name and the provided port number. Valid port numbers must be within the 1024 - 65535 range.

/? Displays usage information for the db2iupdt command.

Examples (UNIX and Linux)

- An instance, `db2inst2`, is associated to a DB2 copy of DB2 database product installed at *DB2DIR1*. You have another DB2 copy of DB2 database product on the same computer at *DB2DIR2* for the same version of the DB2 database product as that installed on *DB2DIR1*. To update the instance to run from the DB2 copy installed at *DB2DIR1* to the DB2 copy installed at *DB2DIR2*, issue the following command:

  ```
  DB2DIR2/instance/db2iupdt db2inst2
  ```

  If the DB2 copy installed at *DB2DIR2* is at level lower than the DB2 copy installed at *DB2DIR1*, issue:

  ```
  DB2DIR2/instance/db2iupdt -D db2inst2
  ```

Usage notes

For all supported operating systems

- If you use the db2iupdt command to update a DB2 instance from one DB2 copy to another DB2 copy of the same version of DB2 database product, the DB2 Global Profile Variables defined in an old DB2 copy installation path will not be updated over to the new installation location. The DB2 Instance Profile Variables specific to the instance will be carried over after the instance is updated.

For UNIX and Linux operating systems

- 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.
- If you want to update a non-root instance, refer to the db2nrupdt non-root-installed instance update command. The db2iupdt does not support updating of non-root instances.
- If you are using the su command instead of the login command to become the root user, you must issue the su command with the **-** option to indicate that the process environment is to be set as if you had logged in to the system using the login command.
- You must not source the DB2 instance environment for the root user. Running db2iupdt when you sourced the DB2 instance environment is not supported.

- On UNIX and Linux operating systems, if the IBM Tivoli Monitoring for Databases: DB2 Agent is installed and the DB2 instance is updated, the Monitoring Agent for DB2 instance is also created if the following are true:
  - The DB2 instance type is standalone, wse, or ese.
  - The DB2 instance is at version 9.5 (or higher).

  In addition, ITMA must already be installed for the DB2 copy you are updating the instance. Located in DB2DIR/itma directory, where DB2DIR represents the directory where the DB2 product is installed.
- On AIX 6.1 (or higher), when running this command from a shared DB2 copy in a system workload partition (WPAR) global environment, this command must be run as the root user.

For Windows operating systems

- The db2iupdt command is located in the *DB2PATH*\bin directory, where *DB2PATH* is the location where the current version of the DB2 database product is installed.
- The instance is updated to the DB2 copy from which you issue the db2iupdt command. However, to move your instance profile from its current location to another location, use the **/p** option and specify the instance profile path. Otherwise, the instance profile will stay in its original location after the instance update. Use the db2iupgrade command instead to upgrade to the current release from a previous release.
- On Windows operating systems, if the IBM Tivoli Monitoring for Databases: DB2 Agent is installed and the DB2 copy instance is updated, the Monitoring Agent for DB2 instance is also created if the following are true:
  - The DB2 instance type is standalone, wse, or ese.
  - The default DB2 copy has the ITM agent component installed.
  - The DB2 instance is at version 9.5 (or higher).
  - There is no existing ITM for Databases product.

  In addition, the following are also created after the creation of the Monitoring Agent for DB2 instance: the Monitoring Agent for DB2 instance files, the NT service and the registry entries.

db2nchg - Change database partition server configuration

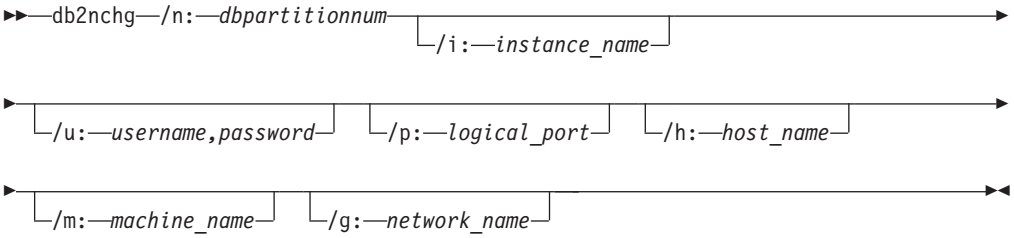
Modifies database partition server configuration. This includes moving the database partition server (node) from one machine to another; changing the TCP/IP host name of the machine; and selecting a different logical port number or a different network name for the database partition server (node). This command can only be used if the database partition server is stopped.

This command is available on Windows operating systems only.

Authorization

Local Administrator

Command syntax



Command parameters

/n:*dbpartitionnum*
Specifies the database partition number of the database partition server's configuration that is to be changed.

/i:*instance_name*
Specifies the instance in which this database partition server participates. If a parameter is not specified, the default is the current instance.

/u:*username,password*
Specifies the user name and password. If a parameter is not specified, the existing user name and password will apply.

/p:*logical_port*
Specifies the logical port for the database partition server. This parameter must be specified to move the database partition server to a different machine. If a parameter is not specified, the logical port number will remain unchanged.

/h:*host_name*
Specifies TCP/IP host name used by FCM for internal communications. If this parameter is not specified, the host name will remain the same.

/m:*machine_name*
Specifies the machine where the database partition server will reside. The database partition server can only be moved if there are no existing databases in the instance.

/g:*network_name*
Changes the network name for the database partition server. This parameter can be used to apply a specific IP address to the database partition server when there are multiple IP addresses on a machine. The network name or the IP address can be entered.

Examples

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

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

db2ncrt - Add database partition server to an instance

Adds a database partition server (node) to an instance.

This command is available on Windows operating systems only.

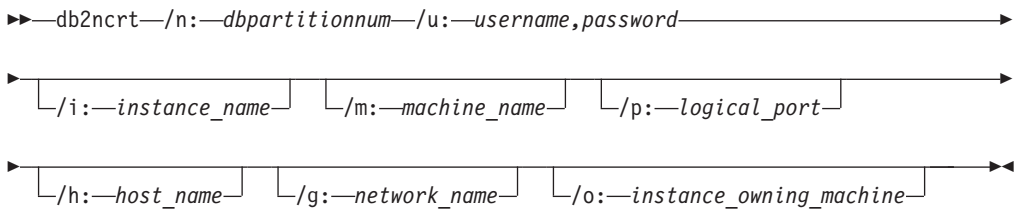
Scope

If a database partition server is added to a computer where an instance already exists, a database partition server is added as a logical database partition server to the computer. If a database partition server is added to a computer where an instance does not exist, the instance is added and the computer becomes a new physical database partition server. This command should not be used if there are databases in an instance. Instead, the START DATABASE MANAGER command should be issued with the ADD DBPARTITIONNUM option. This ensures that the database is correctly added to the new database partition server. It is also possible to add a database partition server to an instance in which a database has been created. The `db2nodes.cfg` file should not be edited since changing the file might cause inconsistencies in the partitioned database environment.

Authorization

`Local Administrator` authority on the computer where the new database partition server is added.

Command syntax



Command parameters

/n:*dbpartitionnum*
A unique database partition number which identifies the database partition server. The number entered can range from 1 to 999.

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

/i:*instance_name*
Specifies the instance name. If a parameter is not specified, the default is the current instance.

/m:*machine_name*
Specifies the computer name of the Windows workstation on which the database partition server resides. This parameter is required if a database partition server is added on a remote computer.

/p:*logical_port*
Specifies the logical port number used for the database partition server. If this parameter is not specified, the logical port number assigned will be 0. When creating a logical database partition server, this parameter must be specified and a logical port number that is not in use must be selected. Note the following restrictions:

- Every computer must have a database partition server that has a logical port 0.
- The port number cannot exceed the port range reserved for FCM communications in the `x:\winnt\system32\drivers\etc\` directory. For

example, if a range of 4 ports is reserved for the current instance, then the maximum port number is 3. Port 0 is used for the default logical database partition server.

/h:*host_name*
Specifies the TCP/IP host name that is used by FCM for internal communications. This parameter is required when the database partition server is being added on a remote computer.

/g:*network_name*
Specifies the network name for the database partition server. If a parameter is not specified, the first IP address detected on the system will be used. This parameter can be used to apply a specific IP address to the database partition server when there are multiple IP addresses on a computer. The network name or the IP address can be entered.

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

Examples

To add a new database partition server to the instance TESTMPP on the instance-owning computer SHAYER, where the new database partition server is known as database partition 2 and uses logical port 1, enter the following command:

```
db2ncrt /n:2 /u:QBPAULZ\paulz,g1reeky /i:TESTMPP /m:TEST /p:1 /o:SHAYER /h:TEST
```

db2ndrop - Drop database partition server from an instance

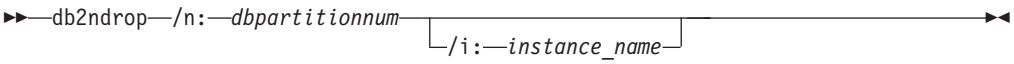
Drops a database partition server (node) from an instance that has no databases. If a database partition server is dropped, its database partition number can be reused for a new database partition server. This command can only be used if the database partition server is stopped.

This command is available on Windows operating systems only.

Authorization

`Local Administrator` authority on the machine where the database partition server is being dropped.

Command syntax



Command parameters

/n:*dbpartitionnum*
A unique database partition number which identifies the database partition server.

/i:*instance_name*
Specifies the instance name. If a parameter is not specified, the default is the current instance.

Examples

```
db2ndrop /n:2 /i=KMASCI
```

Usage notes

If the instance-owning database partition server (dbpartitionnum 0) is dropped from the instance, the instance becomes unusable. To drop the instance, use the db2idrop command.

This command should not be used if there are databases in this instance. Instead, the `db2stop drop nodenum` command should be used. This ensures that the database partition server is correctly removed from the partition database environment. It is also possible to drop a database partition server in an instance where a database exists. The `db2nodes.cfg` file should not be edited since changing the file might cause inconsistencies in the partitioned database environment.

To drop a database partition server that is assigned to the logical port 0 from a machine that is running multiple logical database partition servers, all other database partition servers assigned to the other logical ports must be dropped first. Each database partition server must have a database partition server assigned to logical port 0.

db2rbind - Rebind all packages

Rebinds packages in a database.

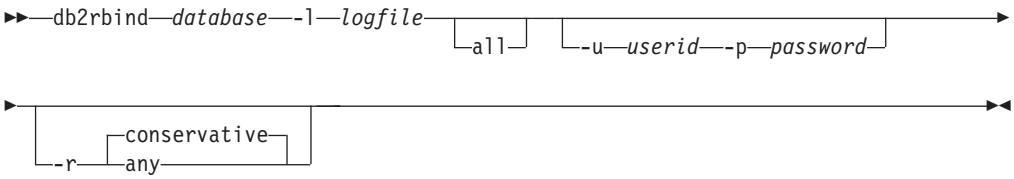
Authorization

dbadm

Required connection

None

Command syntax



Command parameters

database
Specifies an alias name for the database whose packages are to be revalidated.

-l *logfile*
Specifies the (optional) path and the (mandatory) file name to be used for recording the package revalidation process.

Example:

```
cat <logfile>
Starting time .... Thu Jun 18 02:47:11 2009
Succeeded to rebind = 0
Failed to rebind = 0
Ending time .... Thu Jun 18 02:47:11 2009
```

all Specifies that rebinding of all valid and invalid packages is to be done. If this option is not specified, all packages in the database are examined, but only those packages that are marked as invalid are rebound, so that they are not rebound implicitly during application execution.

-u *userid*
User ID. This parameter must be specified if a password is specified.

-p *password*
Password. This parameter must be specified if a user ID is specified.

-r Resolve. Specifies whether rebinding of the package is to be performed with or without conservative binding semantics. This affects whether new objects that use the SQL path for resolution are considered during resolution on static DML statements in the package. This option is not supported by DRDA. Valid values are:

conservative
Only those objects in the SQL path that were defined before the last explicit bind time stamp are considered for resolving references to any objects that use the SQL path for object resolution. Conservative binding semantics are used. This is the default. This option is not supported for an inoperative package.

any All possible matches in the SQL path are considered for resolving references to any objects that use the SQL path for object resolution. Conservative binding semantics are not used.

Usage notes

- This command uses the rebind API (sqlarbnd) to attempt the re-validation of all packages in a database.
- Use of db2rbind is not mandatory.
- For packages that are invalid, you can choose to allow package revalidation to occur implicitly when the package is first used. You can choose to selectively revalidate packages with either the REBIND or the BIND command.
- If the rebind of any of the packages encounters a deadlock or a lock timeout the rebind of all the packages will be rolled back.

db2extsec - Set permissions for DB2 objects

Sets the permissions for DB2 objects (for example, files, directories, network shares, registry keys and services) on updated DB2 database system installations.

As of DB2 version 9 Fixpak 2, domain groups can be used for extended security.

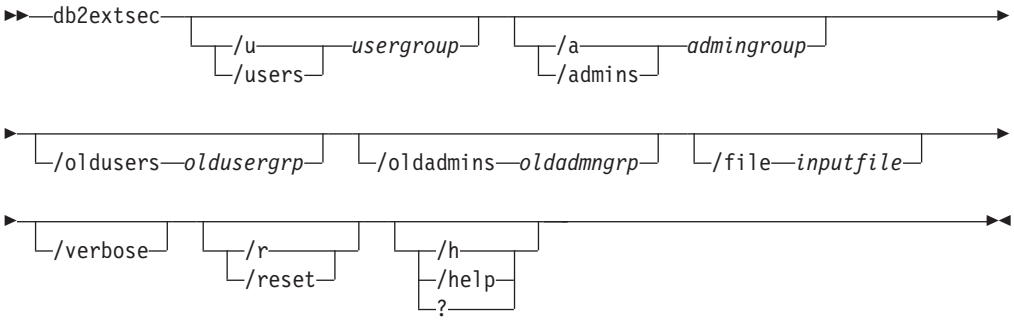
Authorization

sysadm

Required connection

None

Command syntax



Command parameters

/u | /users *usergroup*
Specifies the name of the user group to be added. If this option is not specified, the default DB2 user group (DB2USERS) is used. The *usergroup* can be a local group or a domain group. To specify a local group, you can specify the group name with or without the machine name. For example, `DB2USERS`, or `MYWKSTN\DB2USERS`. To specify a domain group, you specify the *usergroup* in the form of `DOMAIN\GROUP`. For example, `MYDOMAIN\DB2USERS`.

/a | /admins *admingroup*
Specifies the name of the administration group to be added. If this option is not specified, the default DB2 administration group (DB2ADMNS) is used. The *admingroup* can be a local group or a domain group. To specify a local group, you can specify the group name with or without the machine name. For example, `DB2ADMNS`, or `MYWKSTN\DB2ADMNS`. To specify a domain group, you specify the *admingroup* in the form of `DOMAIN\GROUP`. For example, `MYDOMAIN\DB2ADMNS`.

Note: The following 3 parameters, /oldusers, /oldadmins, and /file, are required when you are changing the extended security group names and have file or directory objects that have been created outside of the default locations (i.e., install directory or database directories). The db2extsec command can only change permissions to a known set of DB2 files. If the user had created private DB2 files with extended security, then the user will need to provide the locations of these file, so the db2extsec command can change the permissions on these files with the new extended security group names. The location of the files are to be supplied in the *inputfile* using the /file option.

/oldusers *oldusergrp*
The old DB2 users group name to be changed.

/oldadmins *oldadmngrp*
The old DB2 admins group name to be changed.

/file *inputfile*
File listing additional files/directories for which the permissions need to be updated.

/verbose
Output extra information.

/r | /reset
Specifies that the changes made by previously running db2extsec should be reversed. If you specify this option, all other options are ignored. This option will only work if no other DB2 commands have been issued since the db2extsec command was issued.

/h | /help | ?
Displays the command help information.

Examples

To enable extended security and use the domain groups `mydom\db2users` and `mydom\db2admns` to protect your DB2 objects:

```
db2extsec /u mydom\db2users /a mydom\db2admns
```

To reset extended security to its previous setting (see /reset option above):

```
db2extsec /reset
```

To enable extended security as above, but also change the security group for the files/directories listed in `c:\mylist.lst` from local group `db2admns` and `db2users` to domain groups `mydom\db2admns` and `mydom\db2users`:

```
db2extsec /users mydom\db2users /admins mydom\db2admns /oldadmins db2admns
 /oldusers db2users /file c:\mylist.lst
```

Note: The format of the input file is as follows:

```
* This is a comment
D:\MYBACKUPDIR
D:\MYEXPORTDIR
D:\MYMISCFILE\myfile.dat

* This is another comment
E:\MYOTHERBACKUPDIR             * These are more comments
E:\MYOTHEREXPORTDIR
```

db2secGenerateInitialCred API - Generate initial credentials

The db2secGenerateInitialCred API obtains the initial GSS-API credentials based on the user ID and password that are passed in.

For Kerberos, this is the ticket-granting ticket (TGT). The credential handle that is returned in pGSSCredHandle parameter is the handle that is used with the gss_init_sec_context API and must be either an INITIATE or BOTH credential. The db2secGenerateInitialCred API is only called when a user ID, and possibly a password are supplied. Otherwise, the DB2 database manager specifies the value GSS_C_NO_CREDENTIAL when calling the gss_init_sec_context API to signify that the default credential obtained from the current login context is to be used.

API and data structure syntax

```
SQL_API_RC ( SQL_API_FN *db2secGenerateInitialCred)
                        ( const char *userid,
                          db2int32 useridlen,
                          const char *usernamespace,
                          db2int32 usernamespacelen,
                          db2int32 usernamespacetype,
                          const char *password,
                          db2int32 passwordlen,
```

```
                                const char *newpassword,
                                db2int32 newpasswordlen,
                                const char *dbname,
                                db2int32 dbnamelen,
                                gss_cred_id_t *pGSSCredHandle,
                                void          **InitInfo,
                                char          **errormsg,
                                db2int32 *errormsglen );
```

db2secGenerateInitialCred API parameters

userid Input. The user ID whose password is to be verified on the database server.

useridlen
Input. Length in bytes of the userid parameter value.

usernamespace
Input. The namespace from which the user ID was obtained.

usernamespacelen
Input. Length in bytes of the usernamespace parameter value.

usernamespacetype
Input. The type of namespace.

password
Input. The password to be verified.

passwordlen
Input. Length in bytes of the password parameter value.

newpassword
Input. A new password if the password is to be changed. If no change is requested, the newpassword parameter is set to NULL. If it is not NULL, the API should validate the old password before setting the password to its new value. The API does not have to honour a request to change the password, but if it does not, it should immediately return with the return value DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED without validating the old password.

newpasswordlen
Input. Length in bytes of the newpassword parameter value.

dbname
Input. The name of the database being connected to. The API is free to ignore this parameter, or the API can return the value DB2SEC_PLUGIN_CONNECTION_DISALLOWED if it has a policy of restricting access to certain databases to users who otherwise have valid passwords.

dbnamelen
Input. Length in bytes of the dbname parameter value.

pGSSCredHandle
Output. Pointer to the GSS-API credential handle.

InitInfo
Output. A pointer to data that is not known to DB2. The plug-in can use this memory to maintain a list of resources that are allocated in the process of generating the credential handle. The DB2 database manager calls the db2secFreeInitInfo API at the end of the authentication process, at which point these resources are freed. If the db2secGenerateInitialCred API does not need to maintain such a list, then it should return NULL.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGenerateInitialCred API execution is not successful.

Note: For this API, error messages should not be created if the return value indicates a bad user ID or password. An error message should only be returned if there is an internal error in the API that prevented it from completing properly.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

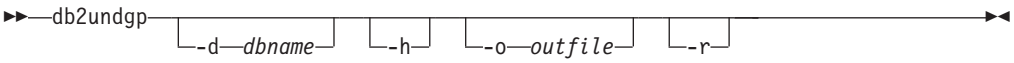
db2undgp - Revoke execute privilege

Note: This command has been deprecated and may be removed in a future release.

Revoke the execute privilege on external stored procedures. This command can be used against external stored procedures.

During the database migration, EXECUTE for all existing functions, methods, and External stored procedures is granted to PUBLIC. This will cause a security exposure for External Stored procedures that contain SQL data access. To prevent users from accessing SQL objects which the user might not have privilege for, use the db2undgp command.

Command syntax



Command parameters

-d *dbname*
Specifies a database name whose maximum length is 8 characters.

-h Displays help for the command.

-o *outfile*
Output the revoke statements in the specified file. Length of the file name should be <= 80.

-r Perform the revoke

Usage notes

At least one of the -r or -o options must be specified.

DROP DATABASE

Deletes the database contents and all log files for the database, uncatalogs the database, and deletes the database subdirectory.

Scope

By default, this command affects all database partitions that are listed in the `db2nodes.cfg` file.

Authorization

One of the following:

- *sysadm*
- *sysctrl*

Required connection

Instance. An explicit attachment is not required. If the database is listed as remote, an instance attachment to the remote database partition server is established for the duration of the command.

Command syntax

Command parameters

DATABASE *database-alias*
Specifies the alias of the database to be dropped. The database must be cataloged in the system database directory.

AT DBPARTITIONNUM
Specifies that the database is to be deleted only on the database partition that issued the DROP DATABASE command. This parameter is used by utilities supplied withDB2 Warehouse Edition and the Database Partitioning Feature (DPF), and is not intended for general use. Improper use of this parameter can cause inconsistencies in the system, so it should only be used with caution.

Examples

The following example deletes the database referenced by the database alias SAMPLE:

```
db2 drop database sample
```

Usage notes

DROP DATABASE deletes all user data and log files, as well as any backup and restore history for the database. If the log files are needed for a roll-forward recovery after a restore operation, or the backup history required to restore the database, these files should be saved prior to issuing this command.

The database must not be in use; all users must be disconnected from the database before the database can be dropped.

To be dropped, a database must be cataloged in the system database directory. Only the specified database alias is removed from the system database directory. If other aliases with the same database name exist, their entries remain. If the

database being dropped is the last entry in the local database directory, the local database directory is deleted automatically.

If DROP DATABASE is issued from a remote client (or from a different instance on the same machine), the specified alias is removed from the client's system database directory. The corresponding database name is removed from the server's system database directory.

Compatibilities

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.

DROP DBPARTITIONNUM VERIFY

Verifies if a database partition exists in the database partition groups of any databases, and if an event monitor is defined on the database partition. This command should be used prior to dropping a database partition from a partitioned database environment.

Scope

This command only affects the database partition on which it is issued.

Authorization

sysadm

Command syntax

Command parameters

None

Usage notes

If a message is returned, indicating that the database partition is not in use, use the STOP DATABASE MANAGER command with DROP DBPARTITIONNUM to remove the entry for the database partition from the `db2nodes.cfg` file, which removes the database partition from the database system.

If a message is returned, indicating that the database partition is in use, the following actions should be taken:

1. If the database partition contains data, redistribute the data to remove it from the database partition using REDISTRIBUTE DATABASE PARTITION GROUP. Use either the DROP DBPARTITIONNUM option on the REDISTRIBUTE DATABASE PARTITION GROUP command or on the ALTER DATABASE PARTITION GROUP statement to remove the database partition from any database partition groups for the database. This must be done for each database that contains the database partition in a database partition group.
2. Drop any event monitors that are defined on the database partition.

3. Rerun DROP DBPARTITIONNUM VERIFY to ensure that the database is no longer in use.

Compatibilities

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.

EXPORT

Exports data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables.

Quick link to "File type modifiers for the export utility" on page 535.

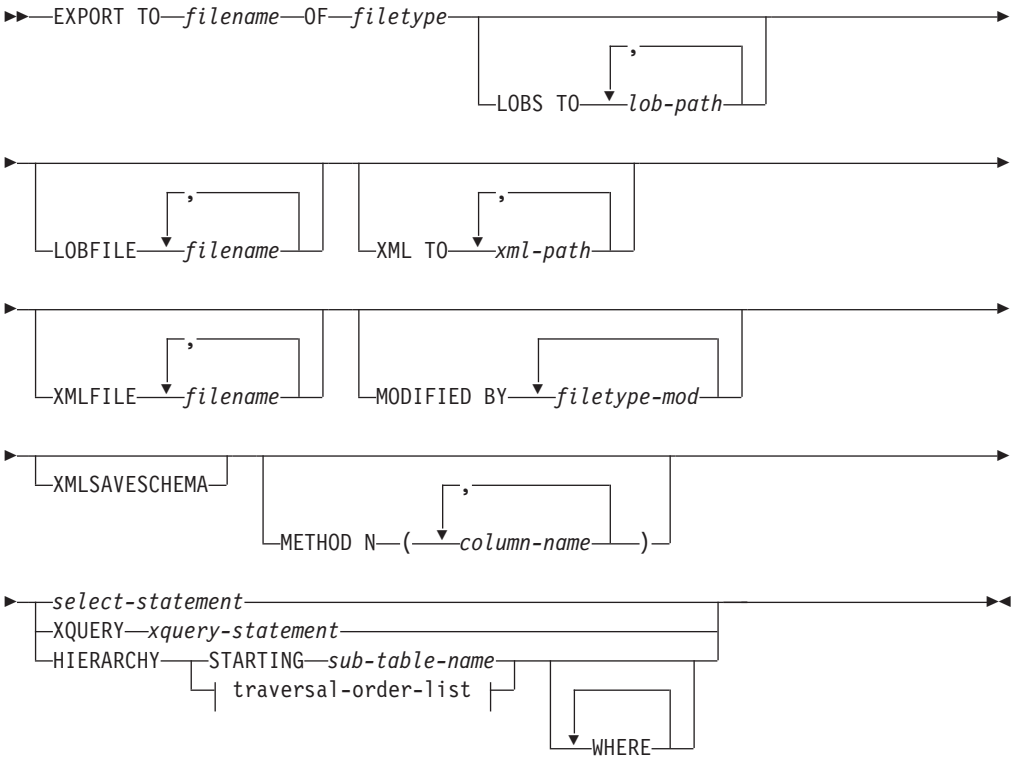
Authorization

One of the following:

- *dataaccess* authority
- CONTROL or SELECT privilege on each participating table or view

Required connection

Command syntax



traversal-order-list:

Command parameters

HIERARCHY *traversal-order-list*
Export a sub-hierarchy using the specified traverse order. All sub-tables must be listed in PRE-ORDER fashion. The first sub-table name is used as the target table name for the SELECT statement.

HIERARCHY STARTING *sub-table-name*
Using the default traverse order (OUTER order for ASC, DEL, or WSF files, or the order stored in PC/IXF data files), export a sub-hierarchy starting from *sub-table-name*.

LOBFILE *filename*
Specifies one or more base file names for the LOB files. When name space is exhausted for the first name, the second name is used, and so on. This will implicitly activate the `LOBSINFILE` behavior.

When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *lob-path*), and then appending a 3-digit sequence number to start and the three character identifier `lob`. For example, if the current LOB path is the directory `/u/foo/lob/path/`, and the current LOB file name is `bar`, the LOB files created will be `/u/foo/lob/path/bar.001.lob`, `/u/foo/lob/path/bar.002.lob`, and so on. The 3-digit sequence number in the LOB file name will grow to 4-digits once 999 is used, 4-digits will grow to 5-digits once 9999 is used, and so on.

LOBS TO *lob-path*
Specifies one or more paths to directories in which the LOB files are to be stored. There will be at least one file per LOB path, and each file will contain at least one LOB. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

METHOD N *column-name*
Specifies one or more column names to be used in the output file. If this parameter is not specified, the column names in the table are used. This parameter is valid only for WSF and IXF files, but is not valid when exporting hierarchical data.

MODIFIED BY *filetype-mod*
Specifies file type modifier options. See "File type modifiers for the export utility" on page 535.

OF *filetype*
Specifies the format of the data in the output file:

- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs.
- WSF (work sheet format), which is used by programs such as:
  - Lotus® 1-2-3®
  - Lotus Symphony

  When exporting BIGINT or DECIMAL data, only values that fall within the range of type DOUBLE can be exported accurately. Although values

that do not fall within this range are also exported, importing or loading these values back might result in incorrect data, depending on the operating system.

Note: Support for the WSF file format is deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed.

- IXF (Integration Exchange Format, PC version) is a proprietary binary format.

select-statement
Specifies the SELECT or XQUERY statement that will return the data to be exported. If the statement causes an error, a message is written to the message file (or to standard output). If the error code is one of SQL0012W, SQL0347W, SQL0360W, SQL0437W, or SQL1824W, the export operation continues; otherwise, it stops.

TO *filename*

If the name of a file that already exists is specified, the export utility overwrites the contents of the file; it does not append the information.

XMLFILE *filename*
Specifies one or more base file names for the XML files. When name space is exhausted for the first name, the second name is used, and so on.

When creating XML files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *xml-path*), appending a 3-digit sequence number, and appending the three character identifier `xml`. For example, if the current XML path is the directory `/u/foo/xml/path/`, and the current XML file name is `bar`, the XML files created will be `/u/foo/xml/path/bar.001.xml`, `/u/foo/xml/path/bar.002.xml`, and so on.

XML TO *xml-path*
Specifies one or more paths to directories in which the XML files are to be stored. There will be at least one file per XML path, and each file will contain at least one XQuery Data Model (XDM) instance. If more than one path is specified, then XDM instances are distributed evenly among the paths.

XMLSAVESCHEMA
Specifies that XML schema information should be saved for all XML columns. For each exported XML document that was validated against an XML schema when it was inserted, the fully qualified SQL identifier of that schema will be stored as an (SCH) attribute inside the corresponding XML Data Specifier (XDS). If the exported document was not validated against an XML schema or the schema object no longer exists in the database, an SCH attribute will not be included in the corresponding XDS.

The schema and name portions of the SQL identifier are stored as the ″OBJECTSCHEMA″ and ″OBJECTNAME″ values in the row of the SYSCAT.XSROBJECTS catalog table corresponding to the XML schema.

The XMLSAVESCHEMA option is not compatible with XQuery sequences that do not produce well-formed XML documents.

Usage notes

- Be sure to complete all table operations and release all locks before starting an export operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.
- Table aliases can be used in the SELECT statement.
- The messages placed in the message file include the information returned from the message retrieval service. Each message begins on a new line.
- The export utility produces a warning message whenever a character column with a length greater than 254 is selected for export to DEL format files.
- PC/IXF import should be used to move data between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.
- The file copying step is not necessary if the source and the target databases are both accessible from the same client.
- DB2 Connect can be used to export tables from DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400®. Only PC/IXF export is supported.
- When exporting to the IXF format, if identifiers exceed the maximum size supported by the IXF format, the export will succeed but the resulting datafile cannot be used by a subsequent import operation using the CREATE mode. SQL27984W will be returned.
- When exporting to a diskette on Windows, and the table that has more data than the capacity of a single diskette, the system will prompt for another diskette, and multiple-part PC/IXF files (also known as multi-volume PC/IXF files, or logically split PC/IXF files), are generated and stored in separate diskettes. In each file, with the exception of the last, there is a DB2 CONTINUATION RECORD (or "AC" Record in short) written to indicate the files are logically split and where to look for the next file. The files can then be transferred to an AIX system, to be read by the import and load utilities. The export utility will not create multiple-part PC/IXF files when invoked from an AIX system. For detailed usage, see the IMPORT command or LOAD command.
- The export utility will store the NOT NULL WITH DEFAULT attribute of the table in an IXF file if the SELECT statement provided is in the form `SELECT * FROM tablename`.
- When exporting typed tables, subselect statements can only be expressed by specifying the target table name and the WHERE clause. Fullselect and *select-statement* cannot be specified when exporting a hierarchy.
- For file formats other than IXF, it is recommended that the traversal order list be specified, because it tells DB2 how to traverse the hierarchy, and what sub-tables to export. If this list is not specified, all tables in the hierarchy are exported, and the default order is the OUTER order. The alternative is to use the default order, which is the order given by the OUTER function.
- Use the same traverse order during an import operation. The load utility does not support loading hierarchies or sub-hierarchies.
- When exporting data from a table that has protected rows, the LBAC credentials held by the session authorization id might limit the rows that are exported. Rows that the session authorization ID does not have read access to will not be exported. No error or warning is given.
- If the LBAC credentials held by the session authorization id do not allow reading from one or more protected columns included in the export then the export fails and an error (SQLSTATE 42512) is returned.

- Export packages are bound using DATETIME ISO format, thus, all date/time/timestamp values are converted into ISO format when cast to a string representation. Since the CLP packages are bound using DATETIME LOC format (locale specific format), you may see inconsistent behavior between CLP and export if the CLP DATETIME format is different from ISO. For instance, the following SELECT statement may return expected results:

```
db2 select col2 from tab1 where char(col2)='05/10/2005';
   COL2
   ----------
   05/10/2005
   05/10/2005
   05/10/2005
   3 record(s) selected.
```

  But an export command using the same select clause will not:

```
db2 export to test.del of del select col2 from test
where char(col2)='05/10/2005';
   Number of rows exported: 0
```

  Now, replacing the LOCALE date format with ISO format gives the expected results:

```
db2 export to test.del of del select col2 from test
where char(col2)='2005-05-10';
   Number of rows exported: 3
```

File type modifiers for the export utility

Table 139. Valid file type modifiers for the export utility: All file formats

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *filename.ext* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string db2exp.001.123.456/ is stored in the data file, the LOB is located at offset 123 in the file db2exp.001, and is 456 bytes long. If you specify the "lobsinfile" modifier when using EXPORT, the LOB data is placed in the locations specified by the LOBS TO clause. Otherwise the LOB data is sent to the data file directory. The LOBS TO clause specifies one or more paths to directories in which the LOB files are to be stored. There will be at least one file per LOB path, and each file will contain at least one LOB. The LOBS TO or LOBFILE options will implicitly activate the LOBSINFILE behavior. To indicate a null LOB , enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
xmlinsepfiles	Each XQuery Data Model (XDM) instance is written to a separate file. By default, multiple values are concatenated together in the same file.
lobsinsepfiles	Each LOB value is written to a separate file. By default, multiple values are concatenated together in the same file.
xmlnodeclaration	XDM instances are written without an XML declaration tag. By default, XDM instances are exported with an XML declaration tag at the beginning that includes an encoding attribute.

Table 139. Valid file type modifiers for the export utility: All file formats (continued)

Modifier	Description
xmlchar	XDM instances are written in the character codepage. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, XDM instances are written out in Unicode.
xmlgraphic	If the `xmlgraphic` modifier is specified with the EXPORT command, the exported XML document will be encoded in the UTF-16 code page regardless of the application code page or the `codepage` file type modifier.

Table 140. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark (″). The specified character is used in place of double quotation marks to enclose a character string.[2] If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:
	`modified by chardel""`
	The single quotation mark (′) can also be specified as a character string delimiter as follows:
	`modified by chardel''`

codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data to this code page from the application code page during the export operation. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. The `codepage` modifier cannot be used with the `lobsinfile` modifier.

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[2]
	In the following example, `coldel;` causes the export utility to use the semicolon character (;) as a column delimiter for the exported data:
	`db2 "export to temp of del modified by coldel;`
	`select * from staff where dept = 20"`

decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[2]
nochardel	Column data will not be surrounded by character delimiters. This option should not be specified if the data is intended to be imported or loaded using DB2. It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.[2]

Table 140. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

striplzeros	Removes the leading zeros from all exported decimal columns.
	
	Consider the following example:
	`db2 create table decimalTable ( c1 decimal( 31, 2 ) )`
	`db2 insert into decimalTable values ( 1.1 )`
	
	`db2 export to data of del select * from decimalTable`
	
	`db2 export to data of del modified by STRIPLZEROS`
	`select * from decimalTable`
	
	In the first export operation, the content of the exported file data will be +0000000000000000000000000000001.10. In the second operation, which is identical to the first except for the `striplzeros` modifier, the content of the exported file data will be +1.10.

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[4] Valid time stamp elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)`
	`MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`U (1 to 12 times) - Fractional seconds(number of occurrences of U represent the number of digits with each digit ranging from 0 to 9`
	`TT - Meridian indicator (AM or PM)`
	
	Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	
	The MMM element will produce the following values: 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', and 'Dec'. 'Jan' is equal to month 1, and 'Dec' is equal to month 12.
	
	The following example illustrates how to export data containing user-defined time stamp formats from a table called 'schedule':
	`db2 export to delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`select * from schedule`

Table 141. Valid file type modifiers for the export utility: IXF file format

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data from this code page to the application code page during the export operation. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive.

Table 142. Valid file type modifiers for the export utility: WSF file format[6]

Modifier	Description
1	Creates a WSF file that is compatible with Lotus 1-2-3 Release 1, or Lotus 1-2-3 Release 1a.[5] This is the default.
2	Creates a WSF file that is compatible with Lotus Symphony Release 1.0.[5]
3	Creates a WSF file that is compatible with Lotus 1-2-3 Version 2, or Lotus Symphony Release 1.1.[5]
4	Creates a WSF file containing DBCS characters.

Note:

1. The export utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the export operation fails, and an error code is returned.
2. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.
3. The export utility normally writes
   - date data in *YYYYMMDD* format
   - char(date) data in *"YYYY-MM-DD"* format
   - time data in *"HH.MM.SS"* format
   - time stamp data in *"YYYY-MM-DD-HH. MM.SS.uuuuuu"* format

   Data contained in any datetime columns specified in the SELECT statement for the export operation will also be in these formats.
4. For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```
5. These files can also be directed to a specific product by specifying an `L` for Lotus 1-2-3, or an `S` for Symphony in the *filetype-mod* parameter string. Only one value or product designator can be specified. Support for the WSF file

format is deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed.

6. The WSF file format is not supported for XML columns. Support for this file format is deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed.
7. All XDM instances are written to XML files that are separate from the main data file, even if neither the XMLFILE nor the XML TO clause is specified. By default, XML files are written to the path of the exported data file. The default base name for XML files is the name of the exported data file with the extension ".xml" appended to it.
8. All XDM instances are written with an XML declaration at the beginning that includes an encoding attribute, unless the XMLNODECLARATION file type modifier is specified.
9. By default, all XDM instances are written in Unicode unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. The default path for XML data and LOB data is the path of the main data file. The default XML file base name is the main data file. The default LOB file base name is the main data file. For example, if the main data file is:

    ```
    /mypath/myfile.del
    ```

    the default path for XML data and LOB data is:

    ```
    /mypath"
    ```

    the default XML file base name is:

    ```
    myfile.del
    ```

    and the default LOB file base name is:

    ```
    myfile.del
    ```

    The LOBSINFILE file type modifier must be specified in order to have LOB files generated.
11. The export utility appends a numeric identifier to each LOB file or XML file. The identifier starts as a 3 digit, 0 padded sequence value, starting at:

    ```
    .001
    ```

    After the 999th LOB file or XML file, the identifier will no longer be padded with zeroes (for example, the 1000th LOG file or XML file will have an extension of:

    ```
    .1000
    ```

    Following the numeric identifier is a three character type identifier representing the data type, either:

    ```
    .lob
    ```

    or

    ```
    .xml
    ```

    For example, a generated LOB file would have a name in the format:

    ```
    myfile.del.001.lob
    ```

    and a generated XML file would be have a name in the format:

    ```
    myfile.del.001.xml
    ```

12. It is possible to have the export utility export XDM instances that are not well-formed documents by specifying an XQuery. However, you will not be able to import or load these exported documents directly into an XML column, since XML columns can only contain complete documents.

GET DATABASE CONFIGURATION

Returns the values of individual entries in a specific database configuration file.

Scope

This command returns information only for the database partition on which it is executed.

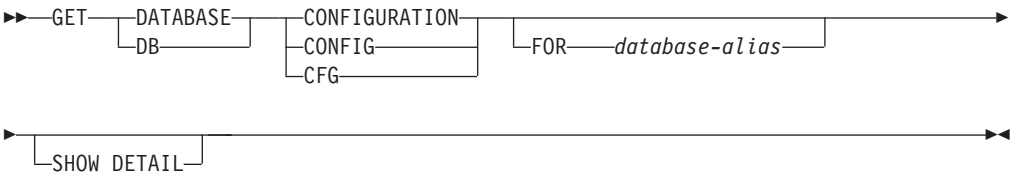
Authorization

None

Required connection

Instance. An explicit attachment is not required, but a connection to the database is required when using the SHOW DETAIL clause. If the database is listed as remote, an instance attachment to the remote node is established for the duration of the command.

Command syntax



Command parameters

FOR *database-alias*
Specifies the alias of the database whose configuration is to be displayed. You do not need to specify the alias if a connection to the database already exists.

SHOW DETAIL
Displays detailed information showing the current value of database configuration parameters as well as the value of the parameters the next time you activate the database. This option lets you see the result of dynamic changes to configuration parameters.

This is a default clause when operating in the CLPPlus interface. SHOW DETAIL need not be called when using CLPPlus processor.

Examples

Note:

1. Output on different platforms might show small variations reflecting platform-specific parameters.

2. Parameters with keywords enclosed by parentheses can be changed by the UPDATE DATABASE CONFIGURATION command.
3. Fields that do not contain keywords are maintained by the database manager and cannot be updated.

The following is sample output from GET DATABASE CONFIGURATION (issued on Windows):

```
     Database Configuration for Database

Database configuration release level                    = 0x0c00
Database release level                                  = 0x0c00

Database territory                                      = US
Database code page                                      = 1252
Database code set                                       = IBM-1252
Database country/region code                            = 1
Database collating sequence                             = UNIQUE
Alternate collating sequence              (ALT_COLLATE) =
Database page size                                      = 8192

Dynamic SQL Query management           (DYN_QUERY_MGMT) = DISABLE

Discovery support for this database       (DISCOVER_DB) = ENABLE

Restrict access                                         = NO
Default query optimization class         (DFT_QUERYOPT) = 5
Degree of parallelism                      (DFT_DEGREE) = 1
Continue upon arithmetic exceptions   (DFT_SQLMATHWARN) = NO
Default refresh age                  (DFT_REFRESH_AGE) = 0
Default maintained table types for opt (DFT_MTTB_TYPES) = SYSTEM
Number of frequent values retained     (NUM_FREQVALUES) = 10
Number of quantiles retained            (NUM_QUANTILES) = 20

Decimal floating point rounding mode  (DECFLT_ROUNDING) = ROUND_HALF_EVEN

Backup pending                                          = NO

All committed transactions have been written to disk    = YES|NO
Rollforward pending                                     = NO
Restore pending                                         = NO

Multi-page file allocation enabled                      = YES

Log retain for recovery status                          = NO
User exit for logging status                            = NO

Self tuning memory                    (SELF_TUNING_MEM) = OFF
Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC
Database memory threshold               (DB_MEM_THRESH) = 10
Max storage for lock list (4KB)               (LOCKLIST) = 50
Percent. of lock lists per application        (MAXLOCKS) = 22
Package cache size (4KB)                    (PCKCACHESZ) = (MAXAPPLS*8)
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = 5000
Sort list heap (4KB)                          (SORTHEAP) = 256

Database heap (4KB)                             (DBHEAP) = AUTOMATIC
Catalog cache size (4KB)              (CATALOGCACHE_SZ) = (MAXAPPLS*4)
Log buffer size (4KB)                           (LOGBUFSZ) = 8
Utilities heap size (4KB)                (UTIL_HEAP_SZ) = 5000
Buffer pool size (pages)                        (BUFFPAGE) = 250
SQL statement heap (4KB)                        (STMTHEAP) = AUTOMATIC
Default application heap (4KB)                (APPLHEAPSZ) = AUTOMATIC
Application Memory Size (4KB)                 (APPL_MEMORY) = AUTOMATIC
Statistics heap size (4KB)                  (STAT_HEAP_SZ) = AUTOMATIC
```

```
Interval for checking deadlock (ms)         (DLCHKTIME) = 10000
Lock timeout (sec)                         (LOCKTIMEOUT) = -1

Changed pages threshold                (CHNGPGS_THRESH) = 60
Number of asynchronous page cleaners   (NUM_IOCLEANERS) = AUTOMATIC
Number of I/O servers                   (NUM_IOSERVERS) = AUTOMATIC
Index sort flag                            (INDEXSORT) = YES
Sequential detect flag                     (SEQDETECT) = YES
Default prefetch size (pages)         (DFT_PREFETCH_SZ) = AUTOMATIC

Track modified pages                        (TRACKMOD) = OFF

Default number of containers                           = 1
Default tablespace extentsize (pages)   (DFT_EXTENT_SZ) = 32

Max number of active applications            (MAXAPPLS) = AUTOMATIC
Average number of active applications       (AVG_APPLS) = AUTOMATIC
Max DB files open per application            (MAXFILOP) = 32768

Log file size (4KB)                         (LOGFILSIZ) = 1000
Number of primary log files                (LOGPRIMARY) = 3
Number of secondary log files               (LOGSECOND) = 2
Changed path to log files                  (NEWLOGPATH) =
Path to log files                                       = D:\DB2\NODE0000\SQL00001\SQLOGDIR\
Overflow log path                     (OVERFLOWLOGPATH) =
Mirror log path                         (MIRRORLOGPATH) =
First active log file                                   =
Block log on disk full                (BLK_LOG_DSK_FUL) = NO
Percent max primary log space by transaction  (MAX_LOG) = 0
Num. of active log files for 1 active UOW(NUM_LOG_SPAN) = 0

Group commit count                          (MINCOMMIT) = 1
Percent log file reclaimed before soft chckpt (SOFTMAX) = 100
Log retain for recovery enabled             (LOGRETAIN) = OFF
User exit for logging enabled                (USEREXIT) = OFF

HADR database role                                      = STANDARD
HADR local host name                  (HADR_LOCAL_HOST) =
HADR local service name                (HADR_LOCAL_SVC) =
HADR remote host name                (HADR_REMOTE_HOST) =
HADR remote service name              (HADR_REMOTE_SVC) =
HADR instance name of remote server  (HADR_REMOTE_INST) =
HADR timeout value                       (HADR_TIMEOUT) = 120
HADR log write synchronization mode     (HADR_SYNCMODE) = NEARSYNC
HADR peer window duration (seconds)  (HADR_PEER_WINDOW) = 0

First log archive method                (LOGARCHMETH1) = OFF
Options for logarchmeth1                 (LOGARCHOPT1) =
Second log archive method               (LOGARCHMETH2) = OFF
Options for logarchmeth2                 (LOGARCHOPT2) =
Failover log archive path               (FAILARCHPATH) =
Number of log archive retries on error   (NUMARCHRETRY) = 5
Log archive retry Delay (secs)         (ARCHRETRYDELAY) = 20
Vendor options                              (VENDOROPT) =

Auto restart enabled                     (AUTORESTART) = ON
Index re-creation time and redo index build  (INDEXREC) = SYSTEM (RESTART)
Log pages during index build            (LOGINDEXBUILD) = OFF
Default number of loadrec sessions    (DFT_LOADREC_SES) = 1
Number of database backups to retain   (NUM_DB_BACKUPS) = 12
Recovery history retention (days)     (REC_HIS_RETENTN) = 366
Auto deletion of recovery objects    (AUTO_DEL_REC_OBJ) = OFF

TSM management class                   (TSM_MGMTCLASS) =
TSM node name                           (TSM_NODENAME) =
TSM owner                                  (TSM_OWNER) =
TSM password                            (TSM_PASSWORD) =
```

```
Automatic maintenance                     (AUTO_MAINT) = ON
  Automatic database backup            (AUTO_DB_BACKUP) = OFF
  Automatic table maintenance          (AUTO_TBL_MAINT) = ON
    Automatic runstats                  (AUTO_RUNSTATS) = ON
    Automatic statistics profiling     (AUTO_STATS_PROF) = OFF
      Automatic profile updates          (AUTO_PROF_UPD) = OFF
    Automatic reorganization               (AUTO_REORG) = OFF

Auto-Revalidation                          (AUTO_REVAL) = DISABLED
Currently Committed                        (CUR_COMMIT) = ON
CHAR output with DECIMAL input         (DEC_TO_CHAR_FMT) = NEW
Enable XML Character operations         (ENABLE_XMLCHAR) = YES
WLM Collection Interval                (WLM_COLLECT_INT) = 0
```

The following example shows a portion of the output of the command when you specify the SHOW DETAIL option. The value in the `Delayed Value` column is the value that will be applied the next time you start the instance.

```
     Database Configuration for Database

Description                                  Parameter   Current Value         Delayed Value
-------------------------------------------------------------------------------------------
Database configuration release level                   = 0x0c00
Database release level                                 = 0x0c00

Database territory                                     = US
Database code page                                     = 1252
Database code set                                      = IBM-1252
Database country/region code                           = 1
Database collating sequence                            = UNIQUE              UNIQUE
Alternate collating sequence              (ALT_COLLATE) =
Database page size                                     = 8192                8192

Dynamic SQL Query management           (DYN_QUERY_MGMT) = DISABLE             DISABLE

Discovery support for this database       (DISCOVER_DB) = ENABLE              ENABLE

Restrict access                                        = NO
Default query optimization class         (DFT_QUERYOPT) = 5                   5
Degree of parallelism                      (DFT_DEGREE) = 1                   1
Continue upon arithmetic exceptions   (DFT_SQLMATHWARN) = NO                  NO
Default refresh age                   (DFT_REFRESH_AGE) = 0                   0
Default maintained table types for opt (DFT_MTTB_TYPES) = SYSTEM              SYSTEM
Number of frequent values retained     (NUM_FREQVALUES) = 10                  10
Number of quantiles retained            (NUM_QUANTILES) = 20                  20

Decimal floating point rounding mode  (DECFLT_ROUNDING) = ROUND_HALF_EVEN     ROUND_HALF_EVEN

Backup pending                                         = NO

Database is consistent                                 = NO
Rollforward pending                                    = NO
Restore pending                                        = NO

Multi-page file allocation enabled                     = YES

Log retain for recovery status                         = NO
User exit for logging status                           = NO

Self tuning memory                    (SELF_TUNING_MEM) = OFF                 OFF
Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC(18096)    AUTOMATIC(18096)
Database memory threshold               (DB_MEM_THRESH) = 10                  10
Max storage for lock list (4KB)               (LOCKLIST) = 50                 50
Percent. of lock lists per application        (MAXLOCKS) = 22                 22
Package cache size (4KB)                   (PCKCACHESZ) = (MAXAPPLS*8)        (MAXAPPLS*8)
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = 5000                5000
```

```
Sort list heap (4KB)                         (SORTHEAP) = 256                  256

Database heap (4KB)                           (DBHEAP) = AUTOMATIC(600)        AUTOMATIC(600)
Catalog cache size (4KB)               (CATALOGCACHE_SZ) = (MAXAPPLS*4)        (MAXAPPLS*4)
Log buffer size (4KB)                         (LOGBUFSZ) = 8                   8
Utilities heap size (4KB)                (UTIL_HEAP_SZ) = 5000                 5000
Buffer pool size (pages)                      (BUFFPAGE) = 250                  250
SQL statement heap (4KB)                      (STMTHEAP) = AUTOMATIC(2048)     AUTOMATIC(2048)
Default application heap (4KB)              (APPLHEAPSZ) = AUTOMATIC(256)      AUTOMATIC(256)
Application Memory Size (4KB)               (APPL_MEMORY) = AUTOMATIC(496)     AUTOMATIC(10000)
Statistics heap size (4KB)                (STAT_HEAP_SZ) = AUTOMATIC(4384)     AUTOMATIC(4384)

Interval for checking deadlock (ms)          (DLCHKTIME) = 10000              10000
Lock timeout (sec)                          (LOCKTIMEOUT) = -1                -1

Changed pages threshold                  (CHNGPGS_THRESH) = 60                 60
Number of asynchronous page cleaners     (NUM_IOCLEANERS) = AUTOMATIC(1)       AUTOMATIC(1)
Number of I/O servers                     (NUM_IOSERVERS) = AUTOMATIC(3)       AUTOMATIC(3)
Index sort flag                               (INDEXSORT) = YES                YES
Sequential detect flag                        (SEQDETECT) = YES                YES
Default prefetch size (pages)           (DFT_PREFETCH_SZ) = AUTOMATIC          AUTOMATIC

Track modified pages                          (TRACKMOD) = NO                  NO

Default number of containers                             = 1                   1
Default tablespace extentsize (pages)     (DFT_EXTENT_SZ) = 32                  32

Max number of active applications              (MAXAPPLS) = AUTOMATIC(40)       AUTOMATIC(40)
Average number of active applications         (AVG_APPLS) = AUTOMATIC(1)        AUTOMATIC(1)
Max DB files open per application              (MAXFILOP) = 32768               32768

Log file size (4KB)                           (LOGFILSIZ) = 1000                1000
Number of primary log files                  (LOGPRIMARY) = 3                   3
Number of secondary log files                 (LOGSECOND) = 2                   2
Changed path to log files                    (NEWLOGPATH) =
Path to log files                                         = D:\DB2\NODE0000    D:\DB2\NODE0000
                                                          \SQL00001\SQLOGDIR\  \SQL00001\SQLOGDIR\
Overflow log path                       (OVERFLOWLOGPATH) =
Mirror log path                           (MIRRORLOGPATH) =
First active log file                                     =
Block log on disk full                  (BLK_LOG_DSK_FUL) = NO                  NO
Percent max primary log space by transaction  (MAX_LOG) = 0                   0
Num. of active log files for 1 active UOW(NUM_LOG_SPAN) = 0                   0
Group commit count                            (MINCOMMIT) = 1                 1
Percent log file reclaimed before soft chckpt (SOFTMAX) = 100                 100
Log retain for recovery enabled               (LOGRETAIN) = OFF               OFF
User exit for logging enabled                  (USEREXIT) = OFF               OFF

HADR database role                                        = STANDARD          STANDARD
HADR local host name                    (HADR_LOCAL_HOST) =
HADR local service name                  (HADR_LOCAL_SVC) =
HADR remote host name                  (HADR_REMOTE_HOST) =
HADR remote service name                (HADR_REMOTE_SVC) =
HADR instance name of remote server    (HADR_REMOTE_INST) =
HADR timeout value                         (HADR_TIMEOUT) = 120               120
HADR log write synchronization mode       (HADR_SYNCMODE) = NEARSYNC          NEARSYNC
HADR peer window duration (seconds)    (HADR_PEER_WINDOW) = 0                 0


First log archive method                  (LOGARCHMETH1) = OFF               OFF
Options for logarchmeth1                   (LOGARCHOPT1) =
Second log archive method                 (LOGARCHMETH2) = OFF               OFF
Options for logarchmeth2                   (LOGARCHOPT2) =
Failover log archive path                 (FAILARCHPATH) =
Number of log archive retries on error     (NUMARCHRETRY) = 5                 5
Log archive retry Delay (secs)           (ARCHRETRYDELAY) = 20                20
Vendor options                                (VENDOROPT) =
```

```
Auto restart enabled                        (AUTORESTART) = ON                   ON
Index re-creation time and redo index build  (INDEXREC) = SYSTEM               SYSTEM (RESTART)
Log pages during index build              (LOGINDEXBUILD) = OFF                OFF
Default number of loadrec sessions     (DFT_LOADREC_SES) = 1                    1
Number of database backups to retain    (NUM_DB_BACKUPS) = 12                   12
Recovery history retention (days)      (REC_HIS_RETENTN) = 366                  366
Auto deletion of recovery objects     (AUTO_DEL_REC_OBJ) = OFF                  OFF

TSM management class                      (TSM_MGMTCLASS) =
TSM node name                              (TSM_NODENAME) =
TSM owner                                     (TSM_OWNER) =
TSM password                               (TSM_PASSWORD) =

Automatic maintenance                        (AUTO_MAINT) = ON                   ON
  Automatic database backup               (AUTO_DB_BACKUP) = OFF                OFF
  Automatic table maintenance             (AUTO_TBL_MAINT) = ON                 ON
    Automatic runstats                     (AUTO_RUNSTATS) = ON                 ON
    Automatic statistics profiling        (AUTO_STATS_PROF) = OFF                OFF
      Automatic profile updates             (AUTO_PROF_UPD) = OFF                OFF
    Automatic reorganization                   (AUTO_REORG) = OFF                OFF
Auto-Revalidation                              (AUTO_REVAL) = DISABLED           DISABLED
Currently Committed                            (CUR_COMMIT) = ON                 ON
CHAR output with DECIMAL input            (DEC_TO_CHAR_FMT) = NEW                NEW
Enable XML Character operations            (ENABLE_XMLCHAR) = ON                 ON
WLM Collection Interval                   (WLM_COLLECT_INT) = 0                  0
```

Usage notes

If an error occurs, the information returned is not valid. If the configuration file is invalid, an error message is returned. The database must be restored from a backup version.

To set the database configuration parameters to the database manager defaults, use the RESET DATABASE CONFIGURATION command.

To retrieve information from all database partitions, use the SYSIBMADM.DBCFG administrative view.

GET DATABASE MANAGER CONFIGURATION

Returns the values of individual entries in the database manager configuration file.

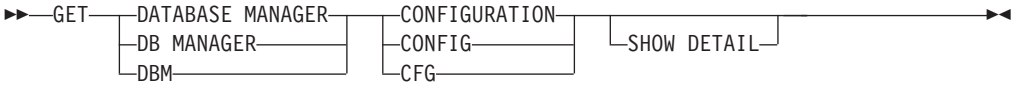
Authorization

None

Required connection

None or instance. An instance attachment is not required to perform local DBM configuration operations, but is required to perform remote DBM configuration operations. To display the database manager configuration for a remote instance, it is necessary to first attach to that instance. The SHOW DETAIL clause requires an instance attachment.

Command syntax



Command parameters

SHOW DETAIL

Displays detailed information showing the current value of database manager configuration parameters as well as the value of the parameters the next time you start the database manager. This option lets you see the result of dynamic changes to configuration parameters.

This is a default clause when operating in the CLPPlus interface. SHOW DETAIL need not be called when using CLPPlus processor.

Examples

Both node type and platform determine which configuration parameters are listed.

The following is sample output from GET DATABASE MANAGER CONFIGURATION (issued on Windows):

```
          Database Manager Configuration

     Node type = Enterprise Server Edition with local and remote clients

 Database manager configuration release level            = 0x0c00

 Maximum total of files open               (MAXTOTFILOP) = 16000
 CPU speed (millisec/instruction)             (CPUSPEED) = 4.251098e-007
 Communications bandwidth (MB/sec)      (COMM_BANDWIDTH) = 1.000000e+002

 Max number of concurrently active databases     (NUMDB) = 8
 Federated Database System Support           (FEDERATED) = NO
 Transaction processor monitor name        (TP_MON_NAME) =

 Default charge-back account           (DFT_ACCOUNT_STR) =

 Java Development Kit installation path       (JDK_PATH) =

 Diagnostic error capture level              (DIAGLEVEL) = 3
 Notify Level                                (NOTIFYLEVEL) = 3
 Diagnostic data directory path                 (DIAGPATH) =

 Default database monitor switches
   Buffer pool                            (DFT_MON_BUFPOOL) = OFF
   Lock                                      (DFT_MON_LOCK) = OFF
   Sort                                      (DFT_MON_SORT) = OFF
   Statement                                 (DFT_MON_STMT) = OFF
   Table                                    (DFT_MON_TABLE) = OFF
   Timestamp                             (DFT_MON_TIMESTAMP) = ON
   Unit of work                                (DFT_MON_UOW) = OFF
 Monitor health of instance and databases   (HEALTH_MON) = ON

 SYSADM group name                          (SYSADM_GROUP) =
 SYSCTRL group name                        (SYSCTRL_GROUP) =
 SYSMAINT group name                      (SYSMAINT_GROUP) =
 SYSMON group name                          (SYSMON_GROUP) =

 Client Userid-Password Plugin           (CLNT_PW_PLUGIN) =
 Client Kerberos Plugin                 (CLNT_KRB_PLUGIN) = IBMkrb5
 Group Plugin                              (GROUP_PLUGIN) =
 GSS Plugin for Local Authorization    (LOCAL_GSSPLUGIN) =
 Server Plugin Mode                    (SRV_PLUGIN_MODE) = UNFENCED
 Server List of GSS Plugins      (SRVCON_GSSPLUGIN_LIST) =
 Server Userid-Password Plugin         (SRVCON_PW_PLUGIN) =
 Server Connection Authentication           (SRVCON_AUTH) = NOT_SPECIFIED
 Cluster manager                            (CLUSTER_MGR) =

 Database manager authentication        (AUTHENTICATION) = SERVER
```

```
Cataloging allowed without authority   (CATALOG_NOAUTH) = NO
Trust all clients                      (TRUST_ALLCLNTS) = YES
Trusted client authentication          (TRUST_CLNTAUTH) = CLIENT
Bypass federated authentication            (FED_NOAUTH) = NO

Default database path                       (DFTDBPATH) = C:

Database monitor heap size (4KB)          (MON_HEAP_SZ) = AUTOMATIC
Java Virtual Machine heap size (4KB)     (JAVA_HEAP_SZ) = 2048
Audit buffer size (4KB)                  (AUDIT_BUF_SZ) = 0
Size of instance shared memory (4KB)  (INSTANCE_MEMORY) = AUTOMATIC
Backup buffer default size (4KB)            (BACKBUFSZ) = 1024
Restore buffer default size (4KB)           (RESTBUFSZ) = 1024

Agent stack size                        (AGENT_STACK_SZ) = 16
Minimum committed private memory (4KB)   (MIN_PRIV_MEM) = 32
Private memory threshold (4KB)        (PRIV_MEM_THRESH) = 20000

Sort heap threshold (4KB)                  (SHEAPTHRES) = 0

Directory cache support                     (DIR_CACHE) = YES

Application support layer heap size (4KB)   (ASLHEAPSZ) = 15
Max requester I/O block size (bytes)         (RQRIOBLK) = 32767
Query heap size (4KB)                   (QUERY_HEAP_SZ) = 1000

Workload impact by throttled utilities(UTIL_IMPACT_LIM) = 10

Priority of agents                           (AGENTPRI) = SYSTEM
Agent pool size                         (NUM_POOLAGENTS) = AUTOMATIC
Initial number of agents in pool        (NUM_INITAGENTS) = 0
Max number of coordinating agents      (MAX_COORDAGENTS) = AUTOMATIC
Max number of client connections       (MAX_CONNECTIONS) = AUTOMATIC

Keep fenced process                        (KEEPFENCED) = YES
Number of pooled fenced processes         (FENCED_POOL) = AUTOMATIC
Initial number of fenced processes     (NUM_INITFENCED) = 0

Index re-creation time and redo index build  (INDEXREC) = RESTART

Transaction manager database name         (TM_DATABASE) = 1ST_CONN
Transaction resync interval (sec)     (RESYNC_INTERVAL) = 180

SPM name                                     (SPM_NAME) = KEON14
SPM log size                           (SPM_LOG_FILE_SZ) = 256
SPM resync agent limit                  (SPM_MAX_RESYNC) = 20
SPM log path                              (SPM_LOG_PATH) =

NetBIOS Workstation name                         (NNAME) =

TCP/IP Service name                         (SVCENAME) = db2c_DB2
Discovery mode                              (DISCOVER) = SEARCH
Discover server instance                (DISCOVER_INST) = ENABLE

Maximum query degree of parallelism   (MAX_QUERYDEGREE) = ANY
Enable intra-partition parallelism     (INTRA_PARALLEL) = NO

Maximum Asynchronous TQs per query    (FEDERATED_ASYNC) = 0

No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = AUTOMATIC
No. of int. communication channels   (FCM_NUM_CHANNELS) = AUTOMATIC
Node connection elapse time (sec)         (CONN_ELAPSE) = 10
Max number of node connection retries (MAX_CONNRETRIES) = 5
Max time difference between nodes (min) (MAX_TIME_DIFF) = 60

db2start/db2stop timeout (min)        (START_STOP_TIME) = 10
```

The following output sample shows the information displayed when you specify the SHOW DETAIL option. The value that appears in the `Delayed Value` column is the value that will be in effect the next time you start the database manager instance.

```
db2 => get dbm cfg show detail

          Database Manager Configuration

     Node type = Enterprise Server Edition with local and remote clients

 Description                                  Parameter   Current Value        Delayed Value
 ------------------------------------------------------------------------------------------
 Database manager configuration release level            = 0x0c00

 Maximum total of files open               (MAXTOTFILOP) = 16000               16000
 CPU speed (millisec/instruction)             (CPUSPEED) = 4.251098e-007       4.251098e-007
 Communications bandwidth (MB/sec)      (COMM_BANDWIDTH) = 1.000000e+002       1.000000e+002

 Max number of concurrently active databases     (NUMDB) = 8                   8
 Federated Database System Support           (FEDERATED) = NO                  NO
 Transaction processor monitor name        (TP_MON_NAME) =

 Default charge-back account           (DFT_ACCOUNT_STR) =

 Java Development Kit installation path       (JDK_PATH) =

 Diagnostic error capture level              (DIAGLEVEL) = 3                   3
 Notify Level                              (NOTIFYLEVEL) = 3                   3
 Diagnostic data directory path               (DIAGPATH) =

 Default database monitor switches
   Buffer pool                         (DFT_MON_BUFPOOL) = OFF                 OFF
   Lock                                   (DFT_MON_LOCK) = OFF                 OFF
   Sort                                   (DFT_MON_SORT) = OFF                 OFF
   Statement                              (DFT_MON_STMT) = OFF                 OFF
   Table                                 (DFT_MON_TABLE) = OFF                 OFF
   Timestamp                        (DFT_MON_TIMESTAMP) = ON                  ON
   Unit of work                           (DFT_MON_UOW) = OFF                 OFF
 Monitor health of instance and databases   (HEALTH_MON) = ON                  ON

 SYSADM group name                         (SYSADM_GROUP) =
 SYSCTRL group name                       (SYSCTRL_GROUP) =
 SYSMAINT group name                     (SYSMAINT_GROUP) =
 SYSMON group name                         (SYSMON_GROUP) =

 Client Userid-Password Plugin          (CLNT_PW_PLUGIN) =

 Client Kerberos Plugin                (CLNT_KRB_PLUGIN) = IBMkrb5             IBMkrb5

 Group Plugin                             (GROUP_PLUGIN) =

 GSS Plugin for Local Authorization    (LOCAL_GSSPLUGIN) =

 Server Plugin Mode                    (SRV_PLUGIN_MODE) = UNFENCED            UNFENCED

 Server List of GSS Plugins      (SRVCON_GSSPLUGIN_LIST) =

 Server Userid-Password Plugin        (SRVCON_PW_PLUGIN) =

 Server Connection Authentication          (SRVCON_AUTH) = NOT_SPECIFIED       NOT_SPECIFIED
 Cluster manager                           (CLUSTER_MGR) =

 Database manager authentication        (AUTHENTICATION) = SERVER              SERVER
 Cataloging allowed without authority   (CATALOG_NOAUTH) = NO                  NO
 Trust all clients                      (TRUST_ALLCLNTS) = YES                 YES
 Trusted client authentication          (TRUST_CLNTAUTH) = CLIENT              CLIENT
```

```
Bypass federated authentication           (FED_NOAUTH) = NO                  NO

Default database path                      (DFTDBPATH) = C:                  C:

Database monitor heap size (4KB)          (MON_HEAP_SZ) = AUTOMATIC(66)      AUTOMATIC(66)
Java Virtual Machine heap size (4KB)     (JAVA_HEAP_SZ) = 2048               2048
Audit buffer size (4KB)                  (AUDIT_BUF_SZ) = 0                  0
Size of instance shared memory (4KB)  (INSTANCE_MEMORY) = AUTOMATIC(73728)   AUTOMATIC(73728)
Backup buffer default size (4KB)           (BACKBUFSZ) = 1024                1024
Restore buffer default size (4KB)          (RESTBUFSZ) = 1024                1024

Agent stack size                        (AGENT_STACK_SZ) = 16                16
Sort heap threshold (4KB)                   (SHEAPTHRES) = 0                 0

Directory cache support                     (DIR_CACHE) = YES                YES

Application support layer heap size (4KB)   (ASLHEAPSZ) = 15                15
Max requester I/O block size (bytes)         (RQRIOBLK) = 32767             32767
Query heap size (4KB)                   (QUERY_HEAP_SZ) = 1000              1000

Workload impact by throttled utilities(UTIL_IMPACT_LIM) = 10                10

Priority of agents                            (AGENTPRI) = SYSTEM            SYSTEM
Agent pool size                         (NUM_POOLAGENTS) = AUTOMATIC(100)    AUTOMATIC(100)
Initial number of agents in pool        (NUM_INITAGENTS) = 0                 0
Max number of coordinating agents      (MAX_COORDAGENTS) = AUTOMATIC(200)    AUTOMATIC(200)
Max number of client connections       (MAX_CONNECTIONS) = AUTOMATIC(MAX_    AUTOMATIC(MAX_
                                                           COORDAGENTS)      COORDAGENTS)

Keep fenced process                          (KEEPFENCED) = YES              YES
Number of pooled fenced processes           (FENCED_POOL) = AUTOMATIC(MAX_   AUTOMATIC(MAX_
                                                           COORDAGENTS)      COORDAGENTS)
Initial number of fenced processes      (NUM_INITFENCED) = 0                 0

Index re-creation time and redo index build  (INDEXREC) = RESTART            RESTART

Transaction manager database name          (TM_DATABASE) = 1ST_CONN          1ST_CONN
Transaction resync interval (sec)      (RESYNC_INTERVAL) = 180               180

SPM name                                      (SPM_NAME) = KEON14            KEON14
SPM log size                             (SPM_LOG_FILE_SZ) = 256             256
SPM resync agent limit                    (SPM_MAX_RESYNC) = 20              20
SPM log path                                (SPM_LOG_PATH) =

NetBIOS Workstation name                          (NNAME) =

TCP/IP Service name                           (SVCENAME) = db2c_DB2          db2c_DB2
Discovery mode                                (DISCOVER) = SEARCH            SEARCH
Discover server instance                 (DISCOVER_INST) = ENABLE            ENABLE

Maximum query degree of parallelism    (MAX_QUERYDEGREE) = ANY               ANY
Enable intra-partition parallelism      (INTRA_PARALLEL) = NO                NO

Maximum Asynchronous TQs per query     (FEDERATED_ASYNC) = 0                 0

No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = AUTOMATIC(4096)    AUTOMATIC(4096)
No. of int. communication channels    (FCM_NUM_CHANNELS) = AUTOMATIC(2048)    AUTOMATIC(2048)
Node connection elapse time (sec)           (CONN_ELAPSE) = 10              10
Max number of node connection retries  (MAX_CONNRETRIES) = 5                 5
Max time difference between nodes (min) (MAX_TIME_DIFF) = 60                60

db2start/db2stop timeout (min)         (START_STOP_TIME) = 10              10
```

Usage notes

- If an attachment to a remote instance or a different local instance exists, the database manager configuration parameters for the attached server are returned; otherwise, the local database manager configuration parameters are returned.
- If an error occurs, the information returned is invalid. If the configuration file is invalid, an error message is returned. The user must drop and recreate the instance to recover.
- To set the configuration parameters to the default values shipped with thedatabase manager , use the RESET DATABASE MANAGER CONFIGURATION command.
- The `AUTOMATIC` values indicated on GET DATABASE MANAGER CONFIGURATION SHOW DETAIL for `FCM_NUM_BUFFERS` and `FCM_NUM_CHANNELS` are the initial values at instance startup time and do not reflect any automatic increasing/decreasing that might have occurred during runtime.
- Configuration parameters **max_connections**, **max_coordagents** and **num_poolagents** are set to AUTOMATIC.
- Configuration parameters **maxagents** and **maxcagents** are deprecated. The following deprecated functions are the result:
  - CLP and the db2CfgSet API will tolerate updates to these parameters, however these updates will be ignored by DB2.
  - CLP will no longer display these database configuration parameters when the client and server are on the DB2 v9.5 code base. If the server is DB2 v9.5, earlier version clients will see a value of 0 output for these parameters. If the client is DB2 v9.5, but the server is prior to DB2 v9.5, these parameters will be displayed with the assigned values.
  - db2CfgGet API will tolerate requests for SQLF_KTN_MAXAGENTS and SQLF_KTN_MAXCAGENTS, but they will return 0 if the server is DB2 v9.5.
  - The behavior of the db2AutoConfig API will depend on the db2VersionNumber passed in to the API. If the version is DB2 v9.5 or beyond, **maxagents** will not be returned, but for versions prior to this it will.
  - The AUTOCONFIGURE CLP command will display a value for **maxagents** with requests from an earlier version client (current and recommended value of 0). For current version client requests, **maxagents** will be displayed with an appropriate value.
  - The AUTOCONFIGURE ADMIN_CMD will not return information about **maxagents** when the server is DB2 v9.5 and beyond.
  - Updates to **maxagents** or **maxcagents** through the ADMIN_CMD will return successfully but have no effect on the server if the server is DB2 v9.5 or later.
  - Queries of database manager configuration parameters using the DBMCFG administrative view will not return rows for **maxagents** or **maxcagents** if the server is DB2 v9.5 or beyond.

  In a future release, these configuration parameters may be removed completely.

IMPORT

Inserts data from an external file with a supported file format into a table, hierarchy, view or nickname. LOAD is a faster alternative, but the load utility does not support loading data at the hierarchy level.

Quick link to “File type modifiers for the import utility” on page 563.

Authorization

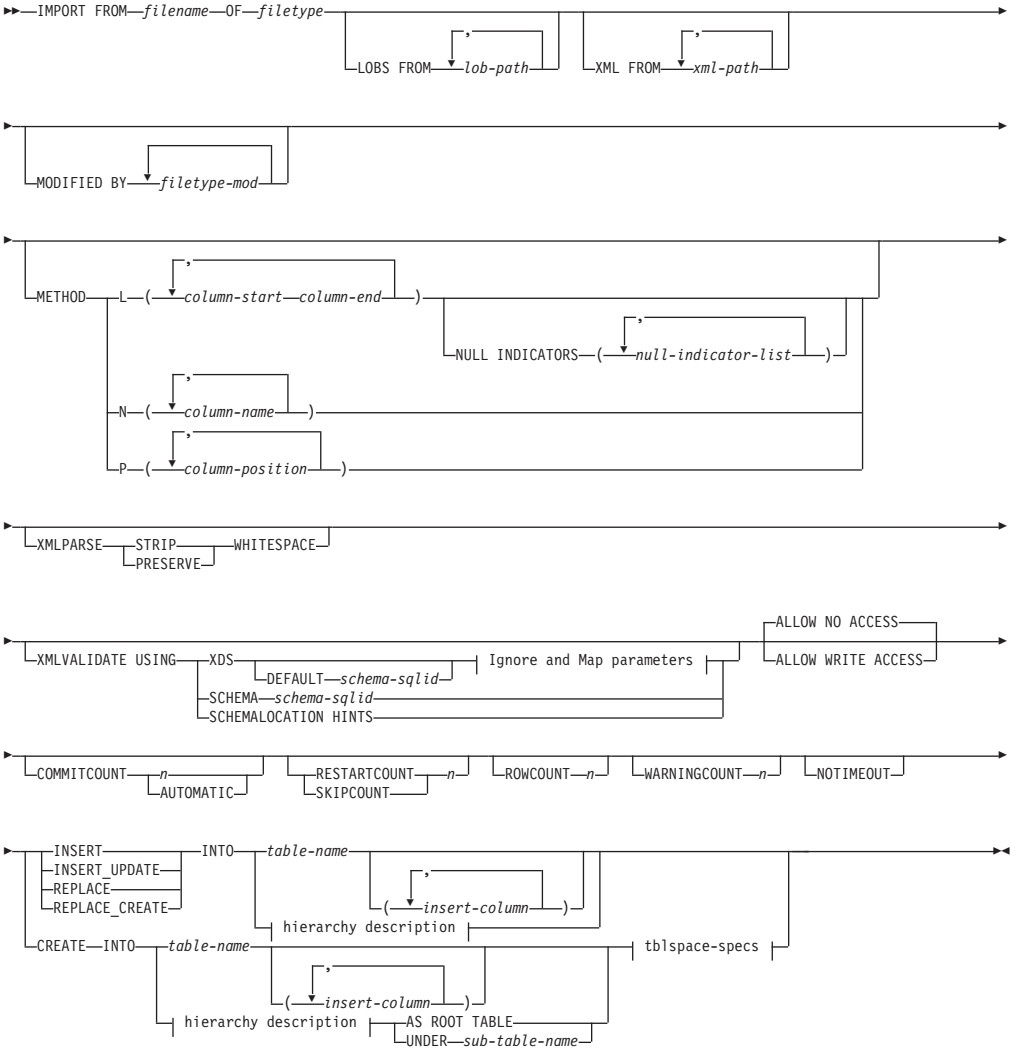
- IMPORT using the **INSERT** option requires one of the following:
  - *dataaccess* authority
  - CONTROL privilege on each participating table, view, or nickname
  - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the **INSERT_UPDATE** option, requires one of the following:
  - *dataaccess* authority
  - CONTROL privilege on each participating table, view, or nickname
  - INSERT, SELECT, UPDATE and DELETE privilege on each participating table or view
- IMPORT to an existing table using the **REPLACE** or **REPLACE_CREATE** option, requires one of the following:
  - *dataaccess* authority
  - CONTROL privilege on the table or view
  - INSERT, SELECT, and DELETE privilege on the table or view
- IMPORT to a new table using the **CREATE** or **REPLACE_CREATE** option, requires one of the following:
  - *dbadm* authority
  - 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
- IMPORT to a hierarchy that does not exist using the **CREATE**, or the **REPLACE_CREATE** option, requires one of the following:
  - *dbadm* authority
  - CREATETAB authority on the database and USE privilege on the table space and one of:
    - IMPLICIT_SCHEMA authority on the database, if the schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema of the table exists
    - CONTROL privilege on every sub-table in the hierarchy, if the **REPLACE_CREATE** option on the entire hierarchy is used
- IMPORT to an existing hierarchy using the **REPLACE** option requires one of the following:
  - *dataaccess* authority
  - CONTROL privilege on every sub-table in the hierarchy
- To import data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the import fails and an error (SQLSTATE 42512) is returned.
- To import data into a table that has protected rows, the session authorization ID must hold LBAC credentials that meet these criteria:
  - It is part of the security policy protecting the table
  - It was granted to the session authorization ID for write access

The label on the row to insert, the user's LBAC credentials, the security policy definition, and the LBAC rules determine the label on the row.

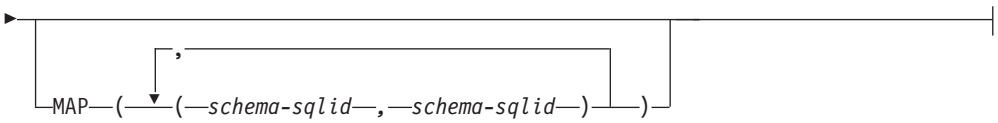
- If the **REPLACE** or **REPLACE_CREATE** option is specified, the session authorization ID must have the authority to drop the table.
- To import data into a nickname, the session authorization ID must have the privilege to access and use a specified data source in pass-through mode.

Required connection

Command syntax

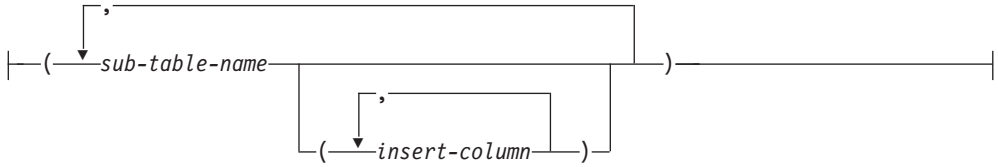


Ignore and Map parameters:



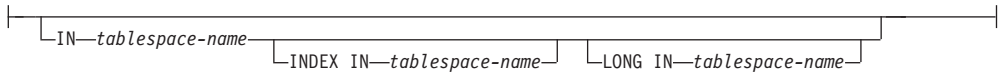
hierarchy description:

sub-table-list:



traversal-order-list:

tblspace-specs:



Command parameters

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the traversal order.

ALLOW NO ACCESS
Runs import in the offline mode. An exclusive (X) lock on the target table is acquired before any rows are inserted. This prevents concurrent applications from accessing table data. This is the default import behavior.

ALLOW WRITE ACCESS
Runs import in the online mode. An intent exclusive (IX) lock on the target table is acquired when the first row is inserted. This allows concurrent readers and writers to access table data. Online mode is not compatible with the **REPLACE**, **CREATE**, or **REPLACE_CREATE** import options. Online mode is not supported in conjunction with buffered inserts. The import operation will periodically commit inserted data to prevent lock escalation to a table lock and to avoid running out of active log space. These commits will be performed even if the **COMMITCOUNT** option was not used. During each commit, import will lose its IX table lock, and will attempt to reacquire it after the commit. This parameter is required

when you import to a nickname and **COMMITCOUNT** must be specified with a valid number (AUTOMATIC is not considered a valid option).

AS ROOT TABLE
Creates one or more sub-tables as a stand-alone table hierarchy.

COMMITCOUNT *n* | **AUTOMATIC**
Performs a COMMIT after every *n* records are imported. When a number *n* is specified, import performs a COMMIT after every *n* records are imported. When compound inserts are used, a user-specified commit frequency of *n* is rounded up to the first integer multiple of the compound count value. When AUTOMATIC is specified, import internally determines when a commit needs to be performed. The utility will commit for either one of two reasons:

- to avoid running out of active log space
- to avoid lock escalation from row level to table level

If the **ALLOW WRITE ACCESS** option is specified, and the **COMMITCOUNT** option is not specified, the import utility will perform commits as if **COMMITCOUNT** AUTOMATIC had been specified.

The ability of the import operation to avoid running out of active log space is affected by the DB2 registry variable **DB2_FORCE_APP_ON_MAX_LOG**:

- If **DB2_FORCE_APP_ON_MAX_LOG** is set to FALSE and the **COMMITCOUNT** AUTOMATIC command option is specified, the import utility will be able to automatically avoid running out of active log space.
- If **DB2_FORCE_APP_ON_MAX_LOG** is set to FALSE and the **COMMITCOUNT** *n* command option is specified, the import utility will attempt to resolve the log full condition if it encounters an SQL0964C (Transaction Log Full) while inserting or updating a record. It will perform an unconditional commit and then will reattempt to insert or update the record. If this does not help resolve the issue (which would be the case when the log full is attributed to other activity on the database), then the IMPORT command will fail as expected, however the number of rows committed may not be a multiple of the **COMMITCOUNT** *n* value. To avoid processing the rows that were already committed when you retry the import operation, use the **RESTARTCOUNT** or **SKIPCOUNT** command parameters.
- If **DB2_FORCE_APP_ON_MAX_LOG** is set to TRUE (which is the default), the import operation will fail if it encounters an SQL0964C while inserting or updating a record. This can occur irrespective of whether you specify **COMMITCOUNT** AUTOMATIC or **COMMITCOUNT** *n*.

  The application is forced off the database and the current unit of work is rolled back. To avoid processing the rows that were already committed when you retry the import operation, use the **RESTARTCOUNT** or **SKIPCOUNT** command parameters.

CREATE

Note: The **CREATE** parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options **CREATE** and **REPLACE_CREATE** are deprecated".

Creates the table definition and row contents in the code page of the database. If the data was exported from a DB2 table, sub-table, or hierarchy, indexes are created. If this option operates on a hierarchy, and data was exported from DB2, a type hierarchy will also be created. This option can only be used with IXF files.

This parameter is not valid when you import to a nickname.

Note: If the data was exported from an MVS™ host database, and it contains LONGVAR fields whose lengths, calculated on the page size, are more than 254, **CREATE** might fail because the rows are too long. See "Imported table re-creation" for a list of restrictions. In this case, the table should be created manually, and IMPORT with **INSERT** should be invoked, or, alternatively, the LOAD command should be used.

DEFAULT *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. The schema specified through the **DEFAULT** clause identifies a schema to use for validation when the XML Data Specifier (XDS) of an imported XML document does not contain an SCH attribute identifying an XML Schema.

The **DEFAULT** clause takes precedence over the **IGNORE** and **MAP** clauses. If an XDS satisfies the **DEFAULT** clause, the **IGNORE** and **MAP** specifications will be ignored.

FROM *filename*

HIERARCHY
Specifies that hierarchical data is to be imported.

IGNORE *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. The **IGNORE** clause specifies a list of one or more schemas to ignore if they are identified by an SCH attribute. If an SCH attribute exists in the XML Data Specifier for an imported XML document, and the schema identified by the SCH attribute is included in the list of schemas to ignore, then no schema validation will occur for the imported XML document.

If a schema is specified in the **IGNORE** clause, it cannot also be present in the left side of a schema pair in the **MAP** clause.

The **IGNORE** clause applies only to the XDS. A schema that is mapped by the **MAP** clause will not be subsequently ignored if specified by the **IGNORE** clause.

IN *tablespace-name*
Identifies the table space in which the table will be created. The table space must exist, and must be a REGULAR table space. If no other table space is specified, all table parts are stored in this table space. If this clause is not specified, the table is created in a table space created by the authorization ID. If none is found, the table is placed into the default table space USERSPACE1. If USERSPACE1 has been dropped, table creation fails.

INDEX IN *tablespace-name*
Identifies the table space in which any indexes on the table will be created. This option is allowed only when the primary table space specified in the **IN** clause is a DMS table space. The specified table space must exist, and must be a REGULAR or LARGE DMS table space.

Note: Specifying which table space will contain an index can only be done when the table is created.

insert-column
Specifies the name of a column in the table or the view into which data is to be inserted.

INSERT
Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE
Adds rows of imported data to the target table, or updates existing rows (of the target table) with matching primary keys.

INTO *table-name*
Specifies the database table into which the data is to be imported. This table cannot be a system table, a created temporary table, a declared temporary table, or a summary table.

One can use an alias for **INSERT**, **INSERT_UPDATE**, or **REPLACE**, except in the case of an earlier server, when the fully qualified or the unqualified table name should be used. A qualified table name is in the form: *schema.tablename*. The *schema* is the user name under which the table was created.

LOBS FROM *lob-path*
The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

This parameter is not valid when you import to a nickname.

LONG IN *tablespace-name*
Identifies the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types, or distinct types with any of these as source types) will be stored. This option is allowed only if the primary table space specified in the **IN** clause is a DMS table space. The table space must exist, and must be a LARGE DMS table space.

MAP *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. Use the **MAP** clause to specify alternate schemas to use in place of those specified by the SCH attribute of an XML Data Specifier (XDS) for each imported XML document. The **MAP** clause specifies a list of one or more schema pairs, where each pair represents a mapping of one schema to another. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

If a schema is present in the left side of a schema pair in the **MAP** clause, it cannot also be specified in the **IGNORE** clause.

Once a schema pair mapping is applied, the result is final. The mapping operation is non-transitive, and therefore the schema chosen will not be subsequently applied to another schema pair mapping.

A schema cannot be mapped more than once, meaning that it cannot appear on the left side of more than one pair.

METHOD

L Specifies the start and end column numbers from which to import data. A column number is a byte offset from the beginning of a row of data. It is numbered starting from 1.

Note: This method can only be used with ASC files, and is the only valid option for that file type.

N Specifies the names of the columns in the data file to be imported. The case of these column names must match the case of the corresponding names in the system catalogs. Each table column that is not nullable should have a corresponding entry in the **METHOD N** list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, method N (F2, F1, F4, F3) is a valid request, while method N (F2, F1) is not valid.

Note: This method can only be used with IXF files.

P Specifies the field numbers of the input data fields to be imported.

Note: This method can only be used with IXF or DEL files, and is the only valid option for the DEL file type.

MODIFIED BY *filetype-mod*

Specifies file type modifier options. See "File type modifiers for the import utility" on page 563.

NOTIMEOUT

Specifies that the import utility will not time out while waiting for locks. This option supersedes the **locktimeout** database configuration parameter. Other applications are not affected.

NULL INDICATORS *null-indicator-list*

This option can only be used when the **METHOD L** parameter is specified. That is, the input file is an ASC file. The null indicator list is a comma-separated list of positive integers specifying the column number of each null indicator field. The column number is the byte offset of the null indicator field from the beginning of a row of data. There must be one entry in the null indicator list for each data field defined in the **METHOD L** parameter. A column number of zero indicates that the corresponding data field always contains data.

A value of `Y` in the NULL indicator column specifies that the column data is NULL. Any character *other than* `Y` in the NULL indicator column specifies that the column data is not NULL, and that column data specified by the **METHOD L** option will be imported.

The NULL indicator character can be changed using the **MODIFIED BY** option, with the `nullindchar` file type modifier.

OF *filetype*

Specifies the format of the data in the input file:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs
- WSF (work sheet format), which is used by programs such as:
  - Lotus 1-2-3
  - Lotus Symphony

- IXF (Integration Exchange Format, PC version) is a binary format that is used exclusively by DB2.

Important: Support for the WSF file format is deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed.

The WSF file type is not supported when you import to a nickname.

REPLACE

Deletes all existing data from the table by truncating the data object, and inserts the imported data. The table definition and the index definitions are not changed. This option can only be used if the table exists. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This parameter is not valid when you import to a nickname.

This option does not honor the CREATE TABLE statement's NOT LOGGED INITIALLY (NLI) clause or the ALTER TABLE statement's ACTIVE NOT LOGGED INITIALLY clause.

If an import with the **REPLACE** option is performed within the same transaction as a CREATE TABLE or ALTER TABLE statement where the NLI clause is invoked, the import will not honor the NLI clause. All inserts will be logged.

Workaround 1

Delete the contents of the table using the DELETE statement, then invoke the import with INSERT statement

Workaround 2

Drop the table and recreate it, then invoke the import with INSERT statement.

This limitation applies to DB2 Universal Database Version 7 and DB2 UDB Version 8

REPLACE_CREATE

Note: The **REPLACE_CREATE** parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options CREATE and REPLACE_CREATE are deprecated".

If the table exists, deletes all existing data from the table by truncating the data object, and inserts the imported data without changing the table definition or the index definitions.

If the table does not exist, creates the table and index definitions, as well as the row contents, in the code page of the database. See *Imported table re-creation* for a list of restrictions.

This option can only be used with IXF files. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This parameter is not valid when you import to a nickname.

RESTARTCOUNT *n*

Specifies that an import operation is to be started at record *n*+1. The first *n* records are skipped. This option is functionally equivalent to **SKIPCOUNT**. **RESTARTCOUNT** and **SKIPCOUNT** are mutually exclusive.

ROWCOUNT *n*

Specifies the number *n* of physical records in the file to be imported (inserted or updated). Allows a user to import only *n* rows from a file, starting from the record determined by the **SKIPCOUNT** or **RESTARTCOUNT** options. If the **SKIPCOUNT** or **RESTARTCOUNT** options are not specified, the first *n* rows are imported. If **SKIPCOUNT** *m* or **RESTARTCOUNT** *m* is specified, rows *m*+1 to *m*+*n* are imported. When compound inserts are used, user specified **ROWCOUNT** *n* is rounded up to the first integer multiple of the compound count value.

SKIPCOUNT *n*

Specifies that an import operation is to be started at record *n*+1. The first *n* records are skipped. This option is functionally equivalent to **RESTARTCOUNT**. **SKIPCOUNT** and **RESTARTCOUNT** are mutually exclusive.

STARTING *sub-table-name*

A keyword for hierarchy only, requesting the default order, starting from *sub-table-name*. For PC/IXF files, the default order is the order stored in the input file. The default order is the only valid order for the PC/IXF file format.

sub-table-list

For typed tables with the **INSERT** or the **INSERT_UPDATE** option, a list of sub-table names is used to indicate the sub-tables into which data is to be imported.

traversal-order-list

For typed tables with the **INSERT**, **INSERT_UPDATE**, or the **REPLACE** option, a list of sub-table names is used to indicate the traversal order of the importing sub-tables in the hierarchy.

UNDER *sub-table-name*

Specifies a parent table for creating one or more sub-tables.

WARNINGCOUNT *n*

Stops the import operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the import file or the target table is specified incorrectly, the import utility will generate a warning for each row that it attempts to import, which will cause the import to fail. If *n* is zero, or this option is not specified, the import operation will continue regardless of the number of warnings issued.

XML FROM *xml-path*

Specifies one or more paths that contain the XML files.

XMLPARSE

Specifies how XML documents are parsed. If this option is not specified, the parsing behavior for XML documents will be determined by the value of the CURRENT XMLPARSE OPTION special register.

STRIP WHITESPACE

Specifies to remove whitespace when the XML document is parsed.

PRESERVE WHITESPACE

Specifies not to remove whitespace when the XML document is parsed.

XMLVALIDATE
Specifies that XML documents are validated against a schema, when applicable.

USING XDS
XML documents are validated against the XML schema identified by the XML Data Specifier (XDS) in the main data file. By default, if the **XMLVALIDATE** option is invoked with the **USING XDS** clause, the schema used to perform validation will be determined by the SCH attribute of the XDS. If an SCH attribute is not present in the XDS, no schema validation will occur unless a default schema is specified by the **DEFAULT** clause.

The **DEFAULT**, **IGNORE**, and **MAP** clauses can be used to modify the schema determination behavior. These three optional clauses apply directly to the specifications of the XDS, and not to each other. For example, if a schema is selected because it is specified by the **DEFAULT** clause, it will not be ignored if also specified by the **IGNORE** clause. Similarly, if a schema is selected because it is specified as the first part of a pair in the MAP clause, it will not be re-mapped if also specified in the second part of another **MAP** clause pair.

USING SCHEMA *schema-sqlid*
XML documents are validated against the XML schema with the specified SQL identifier. In this case, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

USING SCHEMALOCATION HINTS
XML documents are validated against the schemas identified by XML schema location hints in the source XML documents. If a schemaLocation attribute is not found in the XML document, no validation will occur. When the **USING SCHEMALOCATION HINTS** clause is specified, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

See examples of the **XMLVALIDATE** option below.

Usage notes

Be sure to complete all table operations and release all locks before starting an import operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

The import utility adds rows to the target table using the SQL INSERT statement. The utility issues one INSERT statement for each row of data in the input file. If an INSERT statement fails, one of two actions result:

- If it is likely that subsequent INSERT statements can be successful, a warning message is written to the message file, and processing continues.
- If it is likely that subsequent INSERT statements will fail, and there is potential for database damage, an error message is written to the message file, and processing halts.

The utility performs an automatic COMMIT after the old rows are deleted during a **REPLACE** or a **REPLACE_CREATE** operation. Therefore, if the system fails, or the

application interrupts the database manager after the table object is truncated, all of the old data is lost. Ensure that the old data is no longer needed before using these options.

If the log becomes full during a **CREATE**, **REPLACE**, or **REPLACE_CREATE** operation, the utility performs an automatic COMMIT on inserted records. If the system fails, or the application interrupts the database manager after an automatic COMMIT, a table with partial data remains in the database. Use the **REPLACE** or the **REPLACE_CREATE** option to rerun the whole import operation, or use **INSERT** with the **RESTARTCOUNT** parameter set to the number of rows successfully imported.

By default, automatic COMMITs are not performed for the **INSERT** or the **INSERT_UPDATE** option. They are, however, performed if the **COMMITCOUNT** parameter is not zero. If automatic COMMITs are not performed, a full log results in a ROLLBACK.

Offline import does not perform automatic COMMITs if any of the following conditions is true:

- the target is a view, not a table
- compound inserts are used
- buffered inserts are used

By default, online import performs automatic COMMITs to free both the active log space and the lock list. Automatic COMMITs are not performed only if a **COMMITCOUNT** value of zero is specified.

Whenever the import utility performs a COMMIT, two messages are written to the message file: one indicates the number of records to be committed, and the other is written after a successful COMMIT. When restarting the import operation after a failure, specify the number of records to skip, as determined from the last successful COMMIT.

The import utility accepts input data with minor incompatibility problems (for example, character data can be imported using padding or truncation, and numeric data can be imported with a different numeric data type), but data with major incompatibility problems is not accepted.

You cannot **REPLACE** or **REPLACE_CREATE** an object table if it has any dependents other than itself, or an object view if its base table has any dependents (including itself). To replace such a table or a view, do the following:

1. Drop all foreign keys in which the table is a parent.
2. Run the import utility.
3. Alter the table to recreate the foreign keys.

If an error occurs while recreating the foreign keys, modify the data to maintain referential integrity.

Referential constraints and foreign key definitions are not preserved when recreating tables from PC/IXF files. (Primary key definitions *are* preserved if the data was previously exported using `SELECT *`.)

Importing to a remote database requires enough disk space on the server for a copy of the input data file, the output message file, and potential growth in the size of the database.

If an import operation is run against a remote database, and the output message file is very long (more than 60 KB), the message file returned to the user on the client might be missing messages from the middle of the import operation. The first 30 KB of message information and the last 30 KB of message information are always retained.

Importing PC/IXF files to a remote database is much faster if the PC/IXF file is on a hard drive rather than on diskettes.

The database table or hierarchy must exist before data in the **ASC**, **DEL**, or **WSF** file formats can be imported; however, if the table does not already exist, IMPORT **CREATE** or IMPORT **REPLACE_CREATE** creates the table when it imports data from a PC/IXF file. For typed tables, IMPORT **CREATE** can create the type hierarchy and the table hierarchy as well.

PC/IXF import should be used to move data (including hierarchical data) between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand. The file copying step is not necessary if the source and the target databases are both accessible from the same client.

The data in ASC and DEL files is assumed to be in the code page of the client application performing the import. PC/IXF files, which allow for different code pages, are recommended when importing data in different code pages. If the PC/IXF file and the import utility are in the same code page, processing occurs as for a regular application. If the two differ, and the **FORCEIN** option is specified, the import utility assumes that data in the PC/IXF file has the same code page as the application performing the import. This occurs even if there is a conversion table for the two code pages. If the two differ, the **FORCEIN** option is not specified, and there is a conversion table, all data in the PC/IXF file will be converted from the file code page to the application code page. If the two differ, the **FORCEIN** option is not specified, and there is no conversion table, the import operation will fail. This applies only to PC/IXF files on DB2 clients on the AIX operating system.

For table objects on an 8 KB page that are close to the limit of 1012 columns, import of PC/IXF data files might cause DB2 to return an error, because the maximum size of an SQL statement was exceeded. This situation can occur only if the columns are of type CHAR, VARCHAR, or CLOB. The restriction does not apply to import of **DEL** or **ASC** files. If PC/IXF files are being used to create a new table, an alternative is use db2look to dump the DDL statement that created the table, and then to issue that statement through the CLP.

DB2 Connect can be used to import data to DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF import (**INSERT** option) is supported. The **RESTARTCOUNT** parameter, but not the **COMMITCOUNT** parameter, is also supported.

When using the **CREATE** option with typed tables, create every sub-table defined in the PC/IXF file; sub-table definitions cannot be altered. When using options other than **CREATE** with typed tables, the traversal order list enables one to specify the traverse order; therefore, the traversal order list must match the one

used during the export operation. For the PC/IXF file format, one need only specify the target sub-table name, and use the traverse order stored in the file.

The import utility can be used to recover a table previously exported to a PC/IXF file. The table returns to the state it was in when exported.

Data cannot be imported to a system table, a created temporary table, a declared temporary table, or a summary table.

Views cannot be created through the import utility.

Importing a multiple-part PC/IXF file whose individual parts are copied from a Windows system to an AIX system is supported. Only the name of the first file must be specified in the IMPORT command. For example, `IMPORT FROM data.ixf OF IXF INSERT INTO TABLE1`. The file `data.002`, etc should be available in the same directory as `data.ixf`.

On the Windows operating system:

- Importing logically split PC/IXF files is not supported.
- Importing bad format PC/IXF or WSF files is not supported.

Security labels in their internal format might contain newline characters. If you import the file using the DEL file format, those newline characters can be mistaken for delimiters. If you have this problem use the older default priority for delimiters by specifying the `delprioritychar` file type modifier in the IMPORT command.

Federated considerations

When using the IMPORT command and the **INSERT**, **UPDATE**, or **INSERT_UPDATE** command parameters, you must ensure that you have CONTROL privilege on the participating nickname. You must ensure that the nickname you want to use when doing an import operation already exists. There are also several restrictions you should be aware of as shown in the IMPORT command parameters section.

Some data sources, such as ODBC, do not support importing into nicknames.

File type modifiers for the import utility

Table 143. Valid file type modifiers for the import utility: All file formats

Modifier	Description
compound=*x*	*x* is a number between 1 and 100 inclusive. Uses nonatomic compound SQL to insert the data, and *x* statements will be attempted each time. If this modifier is specified, and the transaction log is not sufficiently large, the import operation will fail. The transaction log must be large enough to accommodate either the number of rows specified by **COMMITCOUNT**, or the number of rows in the data file if **COMMITCOUNT** is not specified. It is therefore recommended that the **COMMITCOUNT** option be specified to avoid transaction log overflow. This modifier is incompatible with **INSERT_UPDATE** mode, hierarchical tables, and the following modifiers: `usedefaults`, `identitymissing`, `identityignore`, `generatedmissing`, and `generatedignore`.

Table 143. Valid file type modifiers for the import utility: All file formats (continued)

Modifier	Description
generatedignore	This modifier informs the import utility that data for all generated columns is present in the data file but should be ignored. This results in all values for the generated columns being generated by the utility. This modifier cannot be used with the `generatedmissing` modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated columns (not even NULLs), and will therefore generate a value for each row. This modifier cannot be used with the `generatedignore` modifier.
identityignore	This modifier informs the import utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with the `identitymissing` modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with the `identityignore` modifier.
lobsinfile	*lob-path* specifies the path to the files containing LOB data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *`filename.ext`* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and `mmm` is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file `db2exp.001`, and is 456 bytes long. The **LOBS FROM** clause specifies where the LOB files are located when the "lobsinfile" modifier is used. The **LOBS FROM** clause will implicitly activate the LOBSINFILE behavior. The **LOBS FROM** clause conveys to the IMPORT utility the list of paths to search for the LOB files while importing the data. To indicate a null LOB, enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
no_type_id	Valid only when importing into a single sub-table. Typical usage is to export data from a regular table, and then to invoke an import operation (using this modifier) to convert the data into a single sub-table.
nodefaults	If a source column for a target table column is not explicitly specified, and the table column is not nullable, default values are not loaded. Without this option, if a source column for one of the target table columns is not explicitly specified, one of the following occurs: • If a default value can be specified for a column, the default value is loaded • If the column is nullable, and a default value cannot be specified for that column, a NULL is loaded • If the column is not nullable, and a default value cannot be specified, an error is returned, and the utility stops processing.
norowwarnings	Suppresses all warnings about rejected rows.

Table 143. Valid file type modifiers for the import utility: All file formats (continued)

Modifier	Description
rowchangetimestampignore	This modifier informs the import utility that data for the row change timestamp column is present in the data file but should be ignored. This results in all ROW CHANGE TIMESTAMP being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with the `rowchangetimestampmissing` modifier.
rowchangetimestampmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the row change timestamp column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This modifier cannot be used with the `rowchangetimestampignore` modifier.
seclabelchar	Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. IMPORT converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W)) is returned. This modifier cannot be specified if the seclabelname modifier is specified, otherwise the import fails and an error (SQLCODE SQL3525N) is returned.
seclabelname	Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. IMPORT will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned. This modifier cannot be specified if the seclabelchar modifier is specified, otherwise the import fails and an error (SQLCODE SQL3525N) is returned. **Note:** If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the striptblanks file type modifier to make sure the spaces are removed.
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (", ,") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. **Note:** For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL. Without this option, if a source column contains no data for a row instance, one of the following occurs: • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 144. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data from this code page to the application code page during the import operation. The following rules apply: • For pure DBCS (graphic) mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • `nullindchar` must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. **Note:** 1. The `codepage` modifier cannot be used with the `lobsinfile` modifier. 2. If data expansion occurs when the code page is converted from the application code page to the database code page, the data might be truncated and loss of data can occur.

dateformat="*x*"	*x* is the format of the date in the source file.[2] Valid date elements are:
	
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 1 - 12; mutually exclusive with M)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`.

Table 144. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timeformat="*x*"	*x* is the format of the time in the source file.[2] Valid time elements are:
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`TT - Meridian indicator (AM or PM)`
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	`"HH:MM:SS"`
	`"HH.MM TT"`
	`"SSSSS"`

Table 144. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[2] Valid time stamp elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)`
	`MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`U (1 to 12 times) - Fractional seconds(number of occurrences of U represent the number of digits with each digit ranging from 0 to 9`
	`TT - Meridian indicator (AM or PM)`
	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 'Jan' is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The valid values for the MMM element include: 'jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov' and 'dec'. These values are case insensitive.
	The following example illustrates how to import data containing user defined date and time formats into a table called schedule:
	`db2 import from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

Table 144. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
usegraphiccodepage	If `usegraphiccodepage` is given, the assumption is made that data being imported into graphic or double-byte character large object (DBCLOB) data fields is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic code page is associated with the character code page. IMPORT determines the character code page through either the `codepage` modifier, if it is specified, or through the code page of the application if the `codepage` modifier is not specified. This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data. **Restrictions** The `usegraphiccodepage` modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The `usegraphiccodepage` modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.
xmlchar	Specifies that XML documents are encoded in the character code page. This option is useful for processing XML documents that are encoded in the specified character code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the character code page, otherwise the row containing the document will be rejected. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, either the documents are encoded in Unicode, or they contain a declaration tag with an encoding attribute.
xmlgraphic	Specifies that XML documents are encoded in the specified graphic code page. This option is useful for processing XML documents that are encoded in a specific graphic code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the graphic code page, otherwise the row containing the document will be rejected. Note that the graphic code page is the graphic component of the value specified by the `codepage` file type modifier, or the graphic component of the application code page if it is not specified. By default, documents are either encoded in Unicode, or they contain a declaration tag with an encoding attribute. **Note:** If the `xmlgraphic` modifier is specified with the IMPORT command, the XML document to be imported must be encoded in the UTF-16 code page. Otherwise, the XML document may be rejected with a parsing error, or it may be imported into the table with data corruption.

Table 145. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format

Modifier	Description
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Table 145. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format (continued)

Modifier	Description
nullindchar=*x*	*x* is a single character. Changes the character denoting a null value to *x*. The default value of *x* is `Y`.[3] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the null indicator character is specified to be the letter `N`, then `n` is also recognized as a null indicator.
reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.

striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept.
	
	In the following example, `striptblanks` causes the import utility to truncate trailing blank spaces:
	`db2 import from myfile.asc of asc`
	`modified by striptblanks`
	`method l (1 10, 12 15) messages msgs.txt`
	`insert into staff`
	
	This option cannot be specified together with `striptnulls`. These are mutually exclusive options. This option replaces the obsolete `t` option, which is supported for earlier compatibility only.

striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. This option replaces the obsolete `padwithzero` option, which is supported for earlier compatibility only.

Table 146. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark (″). The specified character is used in place of double quotation marks to enclose a character string.[34] If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:
	`modified by chardel""`
	
	The single quotation mark (′) can also be specified as a character string delimiter. In the following example, `chardel''` causes the import utility to interpret any single quotation mark (′) it encounters as a character string delimiter:
	`db2 "import from myfile.del of del`
	`modified by chardel''`
	`method p (1, 4) insert into staff (id, years)"`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[34]
	
	In the following example, `coldel;` causes the import utility to interpret any semicolon (;) it encounters as a column delimiter:
	`db2 import from myfile.del of del`
	`modified by coldel;`
	`messages msgs.txt insert into staff`

decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.

Table 146. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[34]
	
	In the following example, `decpt;` causes the import utility to interpret any semicolon (;) it encounters as a decimal point:
	`db2 "import from myfile.del of del`
	`modified by chardel''`
	`decpt; messages msgs.txt insert into staff"`

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 import ... modified by delprioritychar ...`
	
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	
	With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and trailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
nochardel	The import utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using DB2 (unless nochardel was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx`, `delprioritychar` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.

Table 147. Valid file type modifiers for the import utility: IXF file format

Modifier	Description
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to import each row.
indexixf	Directs the utility to drop all indexes currently defined on the existing table, and to create new ones from the index definitions in the PC/IXF file. This option can only be used when the contents of a table are being replaced. It cannot be used with a view, or when a *insert-column* is specified.

Table 147. Valid file type modifiers for the import utility: IXF file format (continued)

Modifier	Description
indexschema=*schema*	Uses the specified *schema* for the index name during index creation. If *schema* is not specified (but the keyword `indexschema` *is* specified), uses the connection user ID. If the keyword is not specified, uses the schema in the IXF file.
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
forcecreate	Specifies that the table should be created with possible missing or limited information after returning SQL3311N during an import operation.

Table 148. IMPORT behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	IMPORT behavior
Absent	Absent	All data in the file is assumed to be in the application code page.
Present	Absent	All data in the file is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when imported into the database if `N` is a single-byte code page.
Absent	Present	Character data in the file is assumed to be in the application code page. Graphic data is assumed to be in the code page of the application graphic data. If the application code page is single-byte, then all data is assumed to be in the application code page. **Warning:** If the application code page is single-byte, graphic data will be corrupted when imported into the database, even if the database contains graphic columns.
Present	Present	Character data is assumed to be in code page `N`. Graphic data is assumed to be in the graphic code page of `N`. If `N` is a single-byte or double-byte code page, then all data is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when imported into the database if `N` is a single-byte code page.

Note:

1. The import utility does not issue a warning if an attempt is made to use unsupported file types with the **MODIFIED BY** option. If this is attempted, the import operation fails, and an error code is returned.
2. Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end

positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

```
"M" (could be a month, or a minute)
"M:M" (Which is which?)
"M:YYYY:M" (Both are interpreted as month.)
"S:M:YYYY" (adjacent to both a time value and a date value)
```

In ambiguous cases, the utility will report an error message, and the operation will fail.

Following are some unambiguous time stamp formats:

```
"M:YYYY" (Month)
"S:M" (Minute)
"M:YYYY:S:M" (Month....Minute)
"M:H:YYYY:M:D" (Minute....Month)
```

Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).

3. Character values provided for the chardel, coldel, or decpt file type modifiers must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```

4. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.
5. The following file type modifiers are not allowed when importing into a nickname:
   - `indexixf`
   - `indexschema`
   - `dldelfiletype`
   - `nodefaults`
   - `usedefaults`
   - `no_type_idfiletype`
   - `generatedignore`
   - `generatedmissing`
   - `identityignore`
   - `identitymissing`
   - `lobsinfile`
6. The **WSF** file format is not supported for XML columns. Support for this file format is also deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed
7. The **CREATE** mode is not supported for XML columns.

8. All XML data must reside in XML files that are separate from the main data file. An XML Data Specifier (XDS) (or a NULL value) must exist for each XML column in the main data file.
9. XML documents are assumed to be in Unicode format or to contain a declaration tag that includes an encoding attribute, unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. Rows containing documents that are not well-formed will be rejected.
11. If the **XMLVALIDATE** option is specified, documents that successfully validate against their matching schema will be annotated with the schema information as they are inserted. Rows containing documents that fail to validate against their matching schema will be rejected. To successfully perform the validation, the privileges held by the user invoking the import must include at least one of the following:
    - DBADM authority
    - USAGE privilege on the XML schema to be used in the validation
12. When importing into a table containing an implicitly hidden row change timestamp column, the implicitly hidden property of the column is not honoured. Therefore, the rowchangetimestampmissing file type modifier *must be* specified in the import command if data for the column is not present in the data to be imported and there is no explicit column list present.

INSPECT

Inspect database for architectural integrity, checking the pages of the database for page consistency. The INSPECT command checks that the structures of table objects and structures of table spaces are valid. Cross object validation conducts an online index to data consistency check.

Scope

In a single partition database environment, the scope is that single partition only. In a partitioned database environment, it is the collection of all logical partitions defined in `db2nodes.cfg`. For partitioned tables, the CHECK DATABASE and CHECK TABLESPACE options include individual data partitions and non-partitioned indexes. The CHECK TABLE option is also available for a partitioned table, however it will check all data partitions and indexes in a table, rather than checking a single data partition or index.

Authorization

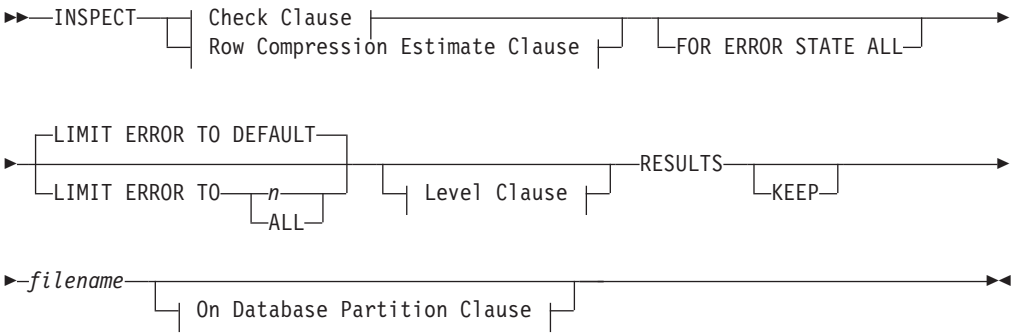
For INSPECT CHECK, one of the following:

- *sysadm*
- *dbadm*
- *sysctrl*
- *sysmaint*
- CONTROL privilege if single table.

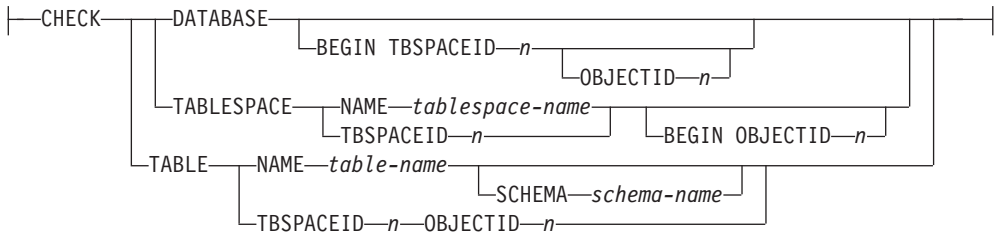
Required Connection

Database

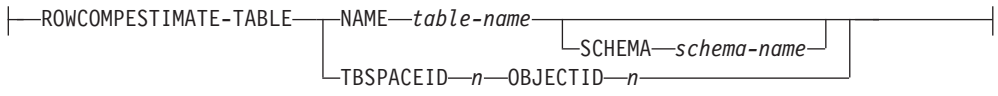
Command Syntax



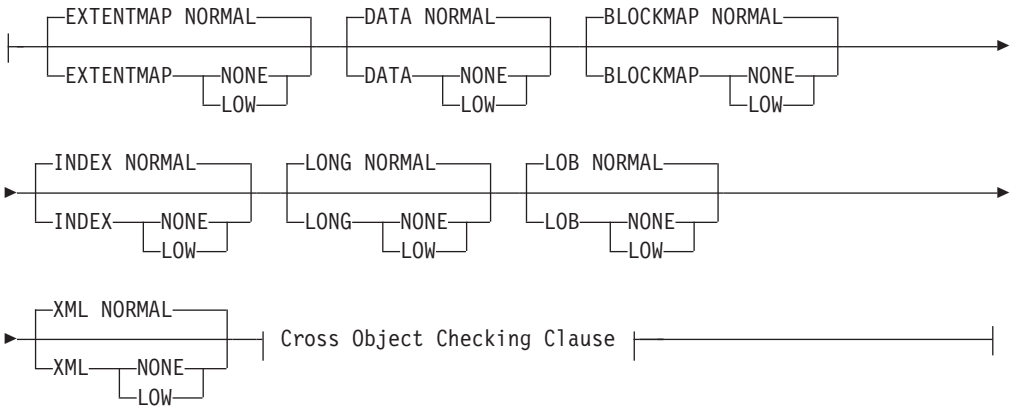
Check Clause:



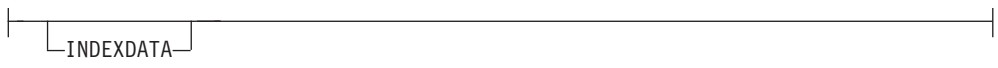
Row Compression Estimate Clause:



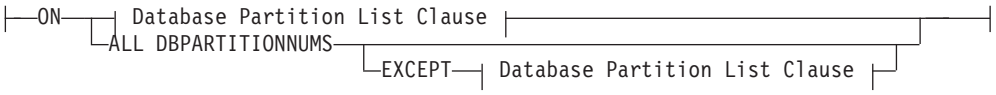
Level Clause:



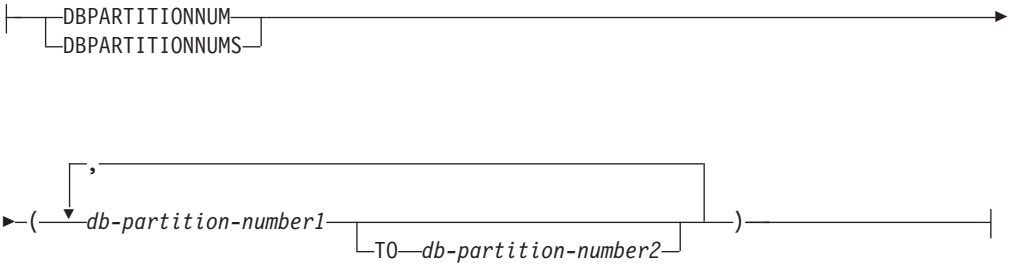
Cross Object Checking Clause:



On Database Partition Clause:



Database Partition List Clause:



Command Parameters

CHECK
Specifies check processing.

DATABASE
Specifies whole database.

BEGIN TBSPACEID *n*
Specifies processing to begin from table space with given table space ID number.

OBJECTID *n*
Specifies processing to begin from table with given table space ID number and object ID number.

TABLESPACE

NAME *tablespace-name*
Specifies single table space with given table space name.

TBSPACEID *n*
Specifies single table space with given table space ID number.

BEGIN OBJECTID *n*
Specifies processing to begin from table with given object ID number.

TABLE

NAME *table-name*
Specifies table with given table name.

SCHEMA *schema-name*
Specifies schema name for specified table name for single table operation.

TBSPACEID *n* **OBJECTID** *n*
Specifies table with given table space ID number and object ID number.

ROWCOMPESTIMATE
Estimates the effectiveness of row compression for a table. You can also specify which database partition(s) this operation is to be done on.

This tool is capable of taking a sample of the table data, and building a dictionary from it. This dictionary can then be used to test compression against the records contained in the sample. From this test compression, data is be gathered from which the following estimates are made:

- Percentage of bytes saved from compression
- Percentage of pages saved from compression
- Compression dictionary size
- Expansion dictionary size

INSPECT will insert the dictionary built for gathering these compression estimates if the `COMPRESS YES` attribute is set for this table, and a dictionary does not already exist for this table. INSPECT will attempt to insert the dictionary concurrent to other applications accessing the table. Dictionary insert requires an Exclusive Table Alter lock and an Intent on Exclusive Table lock. INSPECT will only insert a dictionary into tables that support data row compression. For partitioned tables, a separate dictionary is built and inserted on each partition.

When sampling table row data and building a compression dictionary for a table, the INSPECT command supports only the table row data in the table object. If the table contains XML columns, data is not sampled and a compression dictionary is not built for the XML data in the XML storage object of the table. Use the table function instead.

The ROWCOMPESTIMATE option does not provide an index compression estimate. Use the table function instead.

RESULTS

Specifies the result output file. The file will be written out to the diagnostic data directory path. If there is no error found by the check processing, this result output file will be erased at the end of the INSPECT operation. If there are errors found by the check processing, this result output file will not be erased at the end of the INSPECT operation.

KEEP Specifies to always keep the result output file.

file-name

Specifies the name for the result output file. The file has to be created in the diagnostic data directory path.

ALL DBPARTITIONNUMS

Specifies that operation is to be done on all database partitions specified in the `db2nodes.cfg` file. This is the default if a node clause is not specified.

EXCEPT

Specifies that operation is to be done on all database partitions specified in the `db2nodes.cfg` file, except those specified in the node list.

ON DBPARTITIONNUM | ON DBPARTITIONNUMS

Perform operation on a set of database partitions.

db-partition-number1

Specifies a database partition number in the database partition list.

db-partition-number2

Specifies the second database partition number, so that all database partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

FOR ERROR STATE ALL

For table object with internal state already indicating error state, the check

will just report this status and not scan through the object. Specifying this option will have the processing scan through the object even if internal state already lists error state.

When used with the INDEXDATA option, as long as the index or data object is in an error state, the online index to data consistency checking will not be performed.

LIMIT ERROR TO *n*
Number of pages in error for an object to which reporting is limited. When this limit of the number of pages in error for an object is reached, the processing will discontinue the check on the rest of the object.

When used with the INDEXDATA option, *n* represents the number of errors to which reporting is limited during the online index to data consistency checking.

LIMIT ERROR TO DEFAULT
Default number of pages to limit error reporting for an object. This value is the extent size of the object. This parameter is the default.

When used with the INDEXDATA option, DEFAULT represents the default number of errors to which reporting is limited during the online index to data consistency checking.

LIMIT ERROR TO ALL
No limit on number of pages in error reported.

When used with the INDEXDATA option, ALL represents no limit on the number of errors reported during the online index to data consistency checking.

EXTENTMAP

NORMAL
Specifies processing level is normal for extent map. Default.

NONE
Specifies processing level is none for extent map.

LOW Specifies processing level is low for extent map.

DATA

NORMAL
Specifies processing level is normal for data object. Default.

NONE
Specifies processing level is none for data object.

LOW Specifies processing level is low for data object.

BLOCKMAP

NORMAL
Specifies processing level is normal for block map object. Default.

NONE
Specifies processing level is none for block map object.

LOW Specifies processing level is low for block map object.

INDEX

NORMAL
Specifies processing level is normal for index object. Default.

NONE
Specifies processing level is none for index object.

LOW Specifies processing level is low for index object.

LONG

NORMAL
Specifies processing level is normal for long object. Default.

NONE
Specifies processing level is none for long object.

LOW Specifies processing level is low for long object.

LOB

NORMAL
Specifies processing level is normal for LOB object. Default.

NONE
Specifies processing level is none for LOB object.

LOW Specifies processing level is low for LOB object.

XML

NORMAL
Specifies processing level is normal for XML column object. Default. Pages of XML object will be checked for most inconsistencies. Actual XML data will not be inspected.

NONE
Specifies processing level is none for XML column object. XML object will not be inspected at all.

LOW Specifies processing level is low for XML column object. Pages of XML object will be checked for some inconsistencies. Actual XML data will not be inspected.

INDEXDATA
Specified in order to perform an index to data consistency check. INDEXDATA checking is not performed by default.

Examples

- To perform an index to data consistency check that allows read/write access to all objects, even the object inspected at the moment, issue the following command:

```
inspect check table name fea3 indexdata results keep fea3high.out
```

- To perform an index to data consistency check that allows read or write access to all objects, including the object that is being currently inspected, issue:

```
INSPECT CHECK TABLE NAME car SCHEMA vps INDEXDATA RESULTS KEEP table1.out
```

- To estimate how much storage space will be saved if the data in a table named EMPLOYEE is compressed, issue:

```
INSPECT ROWCOMPESTIMATE TABLE NAME car SCHEMA vps RESULTS table2.out
```

Usage Notes

1. For CHECK operations on table objects, the level of processing can be specified for the objects. The default is NORMAL level, specifying NONE for an object excludes it. Specifying LOW will do subset of checks that are done for NORMAL.

2. The CHECK DATABASE option can be specified to start from a specific table space or from a specific table by specifying the ID value to identify the table space or the table.
3. The CHECK TABLESPACE option can be specified to start from a specific table by specifying the ID value to identify the table.
4. The processing of table spaces will affect only the objects that reside in the table space. The exception is when the INDEXDATA option is used. INDEXDATA will check index to data consistency as long as the index object resides in the table space. This means:
   - If the data object resides in a different table space than the specified table space to be inspected where the index object resides, it can still benefit from the INDEXDATA checking.
   - For a partitioned table, each index can reside in a different table space. Only those indexes that reside in the specified table space will benefit from the index to data checking. If you want to inspect all the indexes against one table, please use the CHECK TABLE option or the CHECK DATABASE option.
5. The online inspect processing will access database objects using isolation level uncommitted read. COMMIT processing will be done during INSPECT processing. It is advisable to end the unit of work by issuing a COMMIT or ROLLBACK before invoking INSPECT.
6. The online inspect check processing will write out unformatted inspection data results to the results file specified. The file will be written out to the diagnostic data directory path. If there is no error found by the check processing, this result output file will be erased at the end of INSPECT operation. If there are errors found by the check processing, this result output file will not be erased at the end of INSPECT operation. After check processing completes, to see inspection details, the inspection result data will require to be formatted out with the utility db2inspf. The results file will have file extension of the database partition number.
7. In a partitioned database environment, each database partition will generate its own results output file with extension corresponding to its database partition number. The output location for the results output file will be the database manager diagnostic data directory path. If the name of a file that already exists is specified, the operation will not be processed, the file will have to be removed before that file name can be specified.
8. Normal online inspect processing will access database objects using isolation level uncommitted read. Inserting a compression dictionary into the table will attempt to acquire write locks. Please refer to the ROWCOMPESTIMATE option for details on dictionary insert locking. Commit processing will be done during the inspect processing. It is advisable to end the unit of work by issuing a COMMIT or ROLLBACK before starting the inspect operation.
9. The INDEXDATA option only examines the logical inconsistency between index and data. Therefore, it is recommended that you first run INDEX and DATA checking separately, to rule out any physical corruption, before running INDEXDATA checking.
10. The INSPECT command, specified with the INDEXDATA parameter, performs an index to data consistency check while allowing read/write access to all objects/tables, even the one being inspected at the moment. The INSPECT INDEXDATA option includes the following inspections:
    - the existence of the data row for a given index entry.
    - a key to data value verification.

    When the INDEXDATA option is specified:

- by default, only the values of explicitly specified level clause options will be used. For any level clause options which are not explicitly specified, the default levels will be overwritten from NORMAL to NONE. For instance, when INDEXDATA is the only level clause option specified, by default, only index to data checking will be performed.

11. The BLOCKMAP option returns information that includes whether a block has been reclaimed for use by the table space following a reorganization to reclaim multidimensional clustering (MDC) table blocks that were empty.

LIST APPLICATIONS

Displays to standard output the application program name, authorization ID (user name), application handle, application ID, and database name of all active database applications. This command can also optionally display an application's sequence number, status, status change time, and database path.

Scope

This command only returns information for the database partition on which it is issued.

Authorization

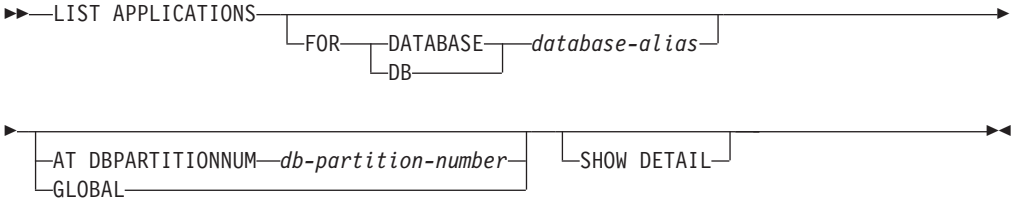
One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT
- SYSMON

Required connection

Instance. To list applications for a remote instance, it is necessary to first attach to that instance.

Command syntax



Command parameters

FOR DATABASE *database-alias*

Information for each application that is connected to the specified database is to be displayed. Database name information is not displayed. If this option is not specified, the command displays the information for each application that is currently connected to any database at the database partition to which the user is currently attached.

The default application information is comprised of the following:

- Authorization ID

- Application name
- Application handle
- Application ID
- Database name
- Number of agents

AT DBPARTITIONNUM *db-partition-number*
Specifies the database partition for which the status of the monitor switches is to be displayed.

GLOBAL
Returns an aggregate result for all database partitions in a partitioned database environment.

SHOW DETAIL
Some of the additional output information will include:

- CONNECT Auth ID
- Sequence number
- Coordinating DB partition number
- Coordinator pid or thread
- Status
- Status change time
- Node
- Database path

If this option is specified, it is recommended that the output be redirected to a file, and that the report be viewed with the help of an editor. The output lines might wrap around when displayed on the screen.

Examples

To list detailed information about the applications connected to the SAMPLE database, issue:

```
list applications for database sample show detail
```

Usage notes

The database administrator can use the output from this command as an aid to problem determination. In addition, this information is required if the database administrator wants to use the GET SNAPSHOT command or the FORCE APPLICATION command in an application.

To list applications at a remote instance (or a different local instance), it is necessary to first attach to that instance. If **FOR DATABASE** is specified when an attachment exists, and the database resides at an instance which differs from the current attachment, the command will fail.

LIST APPLICATIONS only shows user applications while LIST APPLICATIONS SHOW DETAIL shows all applications including the system applications. Event monitors are an example of system applications. System applications usually appear in snapshot output with application names beginning ″db2″ (for example, db2stmm, db2taskd).

Compatibilities

For compatibility with versions earlier than Version 8:

- The keyword **NODE** can be substituted for **DBPARTITIONNUM**.

LIST DATABASE PARTITION GROUPS

Lists all database partition groups associated with the current database.

Scope

This command can be issued from any database partition that is listed in `$HOME/sqllib/db2nodes.cfg`. It returns the same information from any of these database partitions.

Authorization

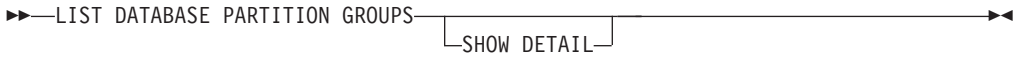
For the system catalogs `SYSCAT.DBPARTITIONGROUPS` and `SYSCAT.DBPARTITIONGROUPDEF`, one of the following is required:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *sysmon*
- *dbadm*
- CONTROL privilege
- SELECT privilege.

Required connection

Database

Command syntax



Command parameters

SHOW DETAIL
Specifies that the output should include the following information:

- Distribution map ID
- Database partition number
- In-use flag

Examples

Following is sample output from the LIST DATABASE PARTITION GROUPS command:

```
DATABASE PARTITION GROUP NAME
-----------------------------
IBMCATGROUP
IBMDEFAULTGROUP

  2 record(s) selected.
```

Following is sample output from the LIST DATABASE PARTITION GROUPS SHOW DETAIL command:

```
DATABASE PARTITION GROUP NAME  PMAP_ID DATABASE PARTITION NUMBER IN_USE
------------------------------ ------- ------------------------ ------
IBMCATGROUP                         0                        0 Y
IBMDEFAULTGROUP                     1                        0 Y

  2 record(s) selected.
```

The fields are identified as follows:

DATABASE PARTITION GROUP NAME
The name of the database partition group. The name is repeated for each database partition in the database partition group.

PMAP_ID
The ID of the distribution map. The ID is repeated for each database partition in the database partition group.

DATABASE PARTITION NUMBER
The number of the database partition.

IN_USE
One of four values:

Y The database partition is being used by the database partition group.

D The database partition is going to be dropped from the database partition group as a result of a REDISTRIBUTE DATABASE PARTITION GROUP operation. When the operation completes, the database partition will not be included in reports from LIST DATABASE PARTITION GROUPS.

A The database partition has been added to the database partition group but is not yet added to the distribution map. The containers for the table spaces in the database partition group have been added on this database partition. The value is changed to `Y` when the REDISTRIBUTE DATABASE PARTITION GROUP operation completes successfully.

T The database partition has been added to the database partition group, but is not yet added to the distribution map. The containers for the table spaces in the database partition group have not been added on this database partition. Table space containers must be added on the new database partition for each table space in the database partition group. The value is changed to `A` when containers have successfully been added.

Compatibilities

For compatibility with versions earlier than Version 8:

- The keyword NODEGROUPS can be substituted for DATABASE PARTITION GROUPS.

LIST PACKAGES/TABLES

Lists packages or tables associated with the current database.

Authorization

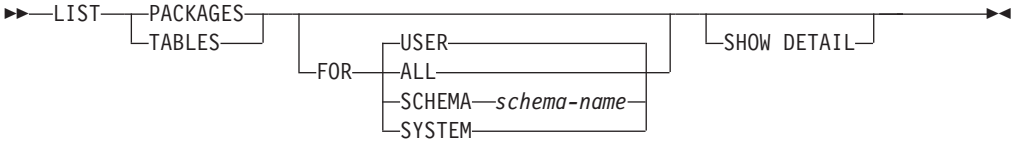
For the system catalog SYSCAT.PACKAGES (LIST PACKAGES) and SYSCAT.TABLES (LIST TABLES), one of the following is required:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *sysmon*
- *dbadm*
- CONTROL privilege
- SELECT privilege.

Required connection

Database. If implicit connect is enabled, a connection to the default database is established.

Command syntax



Command parameters

FOR If the FOR clause is not specified, the packages or tables for USER are listed.

ALL Lists all packages or tables in the database.

SCHEMA *schema-name*
Lists all packages or tables in the database for the specified schema only.

SYSTEM
Lists all system packages or tables in the database.

USER Lists all user packages or tables in the database for the current user.

SHOW DETAIL
If this option is chosen with the LIST TABLES command, the full table name and schema name are displayed. If this option is not specified, the table name is truncated to 30 characters, and the ″>″ symbol in the 31st column represents the truncated portion of the table name; the schema name is truncated to 14 characters and the ″>″ symbol in the 15th column represents the truncated portion of the schema name. If this option is chosen with the LIST PACKAGES command, the full package schema (creator), version and bound by authid are displayed, and the package unique_id (consistency token shown in hexadecimal form). If this option is

not specified, the schema name and bound by ID are truncated to 8 characters and the ″>″ symbol in the 9th column represents the truncated portion of the schema or bound by ID; the version is truncated to 10 characters and the ″>″ symbol in the 11th column represents the truncated portion of the version.

Examples

The following is sample output from LIST PACKAGES:

```
                                Bound     Total                      Isolation
Package    Schema    Version    by        sections     Valid  Format  level     Blocking
---------- --------- ---------- --------- ------------ ------ ------- --------- --------
F4INS      USERA     VER1       SNOWBELL           221 Y      0       CS        U
F4INS      USERA     VER2.0     SNOWBELL           201 Y      0       RS        U
F4INS      USERA     VER2.3     SNOWBELL           201 N      3       CS        U
F4INS      USERA     VER2.5     SNOWBELL           201 Y      0       CS        U
PKG12      USERA                USERA               12 Y      3       RR        B
PKG15      USERA                USERA               42 Y      3       RR        B
SALARY     USERT     YEAR2000   USERT               15 Y      3       CS        N
```

The following is sample output from LIST TABLES:

```
Table/View      Schema          Type       Creation time
--------------- --------------- ---------- --------------------------
DEPARTMENT      SMITH           T          1997-02-19-13.32.25.971890
EMP_ACT         SMITH           T          1997-02-19-13.32.27.851115
EMP_PHOTO       SMITH           T          1997-02-19-13.32.29.953624
EMP_RESUME      SMITH           T          1997-02-19-13.32.37.837433
EMPLOYEE        SMITH           T          1997-02-19-13.32.26.348245
ORG             SMITH           T          1997-02-19-13.32.24.478021
PROJECT         SMITH           T          1997-02-19-13.32.29.300304
SALES           SMITH           T          1997-02-19-13.32.42.973739
STAFF           SMITH           T          1997-02-19-13.32.25.156337

  9 record(s) selected.
```

Usage notes

LIST PACKAGES and LIST TABLES commands are available to provide a quick interface to the system tables.

The following SELECT statements return information found in the system tables. They can be expanded to select the additional information that the system tables provide.

```
select tabname, tabschema, type, create_time
from syscat.tables
order by tabschema, tabname;

select pkgname, pkgschema, pkgversion, unique_id, boundby, total_sect,
   valid, format, isolation, blocking
from syscat.packages
order by pkgschema, pkgname, pkgversion;

select tabname, tabschema, type, create_time
from syscat.tables
where tabschema = 'SYSCAT'
order by tabschema, tabname;

select pkgname, pkgschema, pkgversion, unique_id, boundby, total_sect,
   valid, format, isolation, blocking
from syscat.packages
where pkgschema = 'NULLID'
```

```
order by pkgschema, pkgname, pkgversion;

select tabname, tabschema, type, create_time
from syscat.tables
where tabschema = USER
order by tabschema, tabname;

select pkgname, pkgschema, pkgversion, unique_id, boundby, total_sect,
   valid, format, isolation, blocking
from syscat.packages
where pkgschema = USER
order by pkgschema, pkgname, pkgversion;
```

LIST TABLESPACE CONTAINERS

Lists containers for the specified table space.

Important: This command or API has been deprecated and might be removed in a future release. You can use the MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions instead which return more information. For more information, see the "LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated" topic in the *What's New for DB2 Version 9.7* book.

The table space snapshot contains all of the information displayed by the LIST TABLESPACE CONTAINERS command.

Scope

This command returns information only for the node on which it is executed.

Authorization

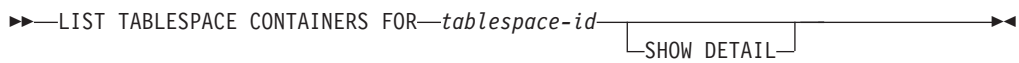
One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *sysmon*
- *dbadm*

Required connection

Database

Command syntax



Command parameters

FOR *tablespace-id*
An integer that uniquely represents a table space used by the current database. To get a list of all the table spaces used by the current database, use the LIST TABLESPACES command.

SHOW DETAIL

If this option is not specified, only the following basic information about each container is provided:

- Container ID
- Name
- Type (file, disk, or path).

If this option is specified, the following additional information about each container is provided:

- Total number of pages
- Number of usable pages
- Accessible (yes or no).

Examples

The following is sample output from `LIST TABLESPACE CONTAINERS FOR 0`:

```
           Tablespace Containers for Tablespace 0

 Container ID              = 0
 Name                      = /home/smith/smith/NODE0000/SQL00001/SQLT0000.0
 Type                      = Path
```

The following is sample output from `LIST TABLESPACE CONTAINERS FOR 0 SHOW DETAIL` specified:

```
           Tablespace Containers for Tablespace 0

 Container ID              = 0
 Name                      = /home/smith/smith/NODE0000/SQL00001/SQLT0000.0
 Type                      = Path
 Total pages               = 895
 Useable pages             = 895
 Accessible                = Yes
```

LIST TABLESPACES

Lists table spaces and information about table spaces for the current database.

Important: This command or API has been deprecated and might be removed in a future release. You can use the MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions instead which return more information. For more information, see the "LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Information displayed by this command is also available in the table space snapshot.

Scope

This command returns information only for the database partition on which it is executed.

Authorization

One of the following:

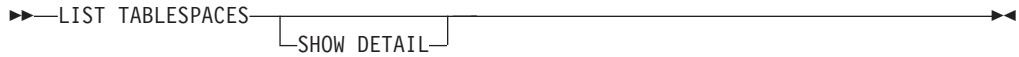
- SYSADM

- SYSCTRL
- SYSMAINT
- SYSMON
- DBADM
- LOAD authority

Required connection

Database

Command syntax



Command parameters

SHOW DETAIL

If this option is not specified, only the following basic information about each table space is provided:

- Table space ID
- Name
- Type (system managed space or database managed space)
- Contents (any data, long or index data, or temporary data)
- State, a hexadecimal value indicating the current table space state. The externally visible state of a table space is composed of the hexadecimal sum of certain state values. For example, if the state is ″quiesced: EXCLUSIVE″ and ″Load pending″, the value is `0x0004` + `0x0008`, which is `0x000c`. The db2tbst (Get Tablespace State) command can be used to obtain the table space state associated with a given hexadecimal value. Following are the bit definitions listed in `sqlutil.h`:

```
0x0          Normal
0x1          Quiesced: SHARE
0x2          Quiesced: UPDATE
0x4          Quiesced: EXCLUSIVE
0x8          Load pending
0x10         Delete pending
0x20         Backup pending
0x40         Roll forward in progress
0x80         Roll forward pending
0x100        Restore pending
0x100        Recovery pending (not used)
0x200        Disable pending
0x400        Reorg in progress
0x800        Backup in progress
0x1000       Storage must be defined
0x2000       Restore in progress
0x4000       Offline and not accessible
0x8000       Drop pending
0x20000      Load in progress
0x2000000    Storage may be defined
0x4000000    StorDef is in 'final' state
0x8000000    StorDef was change prior to roll forward
0x10000000   DMS rebalance in progress
0x20000000   Table space deletion in progress
0x40000000   Table space creation in progress
```

If this option is specified, the following additional information about each table space is provided:

- Total number of pages
- Number of usable pages
- Number of used pages
- Number of free pages
- High water mark (in pages)
- Page size (in bytes)
- Extent size (in pages)
- Prefetch size (in pages)
- Number of containers
- Minimum recovery time (displayed only if not zero)
- State change table space ID (displayed only if the table space state is ″load pending″ or ″delete pending″)
- State change object ID (displayed only if the table space state is ″load pending″ or ″delete pending″)
- Number of quiescers (displayed only if the table space state is ″quiesced: SHARE″, ″quiesced: UPDATE″, or ″quiesced: EXCLUSIVE″)
- Table space ID and object ID for each quiescer (displayed only if the number of quiescers is greater than zero).

Examples

The following are two sample outputs from LIST TABLESPACES SHOW DETAIL.

```
          Tablespaces for Current Database
Tablespace ID                        = 0
Name                                 = SYSCATSPACE
Type                                 = Database managed space
Contents                             = Any data
State                                = 0x0000
  Detailed explanation:
    Normal
Total pages                          = 895
Useable pages                        = 895
Used pages                           = 895
Free pages                           = Not applicable
High water mark (pages)              = Not applicable
Page size (bytes)                    = 4096
Extent size (pages)                  = 32
Prefetch size (pages)                = 32
Number of containers                 = 1

Tablespace ID                        = 1
Name                                 = TEMPSPACE1
Type                                 = System managed space
Contents                             = Temporary data
State                                = 0x0000
  Detailed explanation:
    Normal
Total pages                          = 1
Useable pages                        = 1
Used pages                           = 1
Free pages                           = Not applicable
High water mark (pages)              = Not applicable
Page size (bytes)                    = 4096
Extent size (pages)                  = 32
Prefetch size (pages)                = 32
```

```
 Number of containers                 = 1

 Tablespace ID                        = 2
 Name                                 = USERSPACE1
 Type                                 = Database managed space
 Contents                             = Any data
 State                                = 0x000c
   Detailed explanation:
     Quiesced: EXCLUSIVE
     Load pending
 Total pages                          = 337
 Useable pages                        = 337
 Used pages                           = 337
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1
 State change tablespace ID           = 2
 State change object ID               = 3
 Number of quiescers                  = 1
   Quiescer 1:
     Tablespace ID                    = 2
     Object ID                        = 3
DB21011I  In a partitioned database server environment, only the table spaces
on the current node are listed.

           Tablespaces for Current Database
 Tablespace ID                        = 0
 Name                                 = SYSCATSPACE
 Type                                 = System managed space
 Contents                             = Any data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1200
 Useable pages                        = 1200
 Used pages                           = 1200
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1

 Tablespace ID                        = 1
 Name                                 = TEMPSPACE1
 Type                                 = System managed space
 Contents                             = Temporary data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1
 Useable pages                        = 1
 Used pages                           = 1
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1

Tablespace ID                         = 2
 Name                                 = USERSPACE1
 Type                                 = System managed space
 Contents                             = Any data
```

```
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1
 Useable pages                        = 1
 Used pages                           = 1
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1

Tablespace ID                         = 3
 Name                                 = DMS8K
 Type                                 = Database managed space
 Contents                             = Any data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 2000
 Useable pages                        = 1952
 Used pages                           = 96
 Free pages                           = 1856
 High water mark (pages)              = 96
 Page size (bytes)                    = 8192
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 2

Tablespace ID                         = 4
 Name                                 = TEMP8K
 Type                                 = System managed space
 Contents                             = Temporary data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1
 Useable pages                        = 1
 Used pages                           = 1
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 8192
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1
DB21011I  In a partitioned database server environment, only the table spaces
on the current node are listed.
```

Usage notes

In a partitioned database environment, this command does not return all the table spaces in the database. To obtain a list of all the table spaces, query SYSCAT.TABLESPACES.

During a table space rebalance, the number of usable pages includes pages for the newly added container, but these new pages are not reflected in the number of free pages until the rebalance is complete. When a table space rebalance is not in progress, the number of used pages plus the number of free pages equals the number of usable pages.

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 149. Supported table space states

State	Hexadecimal state value	Description	Examples

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 **logretain** database configuration parameter is set to RECOVERY, or the **userexit** database configuration parameter is set to YES. 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.
			1. Given load input file staff_data.del with content:
			
			11,"Melnyk",20,"Sales",10,70000,15000:
			`update db cfg for sample using logretain recovery;`
			`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;`
			
			2.
			`update db cfg for sample using logretain recovery;`
			`connect to sample;`

Table 149. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Backup in Progress	0x800	This is a transient state that is only in effect during a backup operation.	Issue an online BACKUP DATABASE command:
			`backup db sample online;`
			
			While the backup operation is running, execute the following script from another session:
			`connect to sample;`
			
			1.
			`list tablespaces show detail;`
			
			or
			
			2.
			`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.	Given load input file `staffdata.del` with a substantial amount of data (for example, 20000 or more records):
			`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.

Table 149. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples
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.

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.	Given load input file `staffdata.del` with a substantial amount of data (for example, 20000 or more records):
			`update db cfg for sample using logretain recovery;`
			`backup db sample;`
			`connect to sample;`
			`create table newstaff like staff;`
			`load from staffdata.del of del insert into newstaff copy no;`
			`connect reset;`
			
			While the load operation is running, execute the following script from another session:
			`connect to sample;`
			`list tablespaces;`
			`connect reset;`
			
			Information returned for USERSPACE1 shows that this table space is in Load in Progress (and Backup Pending) state.

Table 149. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

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.	`connect to sample;`
			`create tablespace ts1 managed by database using`
			`(file '/home/melnyk/melnyk/NODE0000/SQL00001`
			`/tsc1' 1024);`
			`list tablespaces show detail;`

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.	`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. You can put a table space in Quiesced Exclusive state explicitly by issuing the QUIESCE TABLESPACES FOR TABLE command.	Ensure that the table space state is Normal before setting it to Quiesced Exclusive.
			`connect to sample;`
			`quiesce tablespaces for table staff reset;`
			`quiesce tablespaces for table staff exclusive;`
			`connect reset;`
			
			Execute the following script from another session:
			`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 149. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

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. You can put a table space in Quiesced Share state explicitly by issuing the QUIESCE TABLESPACES FOR TABLE command.	Ensure that the table space state is Normal before setting it to Quiesced Share.
			`connect to sample;`
			`quiesce tablespaces for table staff reset;`
			`quiesce tablespaces for table staff share;`
			`connect reset;`
			
			Execute the following script from another session:
			`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. You can put a table space in Quiesced Update state explicitly by issuing the QUIESCE TABLESPACES FOR TABLE command.	Ensure that the table space state is Normal before setting it to Quiesced Update.
			`connect to sample;`
			`quiesce tablespaces for table staff reset;`
			`quiesce tablespaces for table staff intent to update;`
			`connect reset;`
			
			Execute the following script from another session:
			`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 149. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Reorg in Progress	0x400	This is a transient state that is only in effect during a reorg operation.	Issue a REORG TABLE command:
			`connect to sample;`
			`reorg table staff;`
			`connect reset;`
			
			While the reorg operation is running, execute the following script from another session:
			`connect to sample;`
			
			1.
			`list tablespaces show detail;`
			
			or
			
			2.
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			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.	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.

Table 149. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Restore in Progress	0x2000	This is a transient state that is only in effect during a restore operation.	`update db cfg for sample using logretain recovery;`
			`backup db sample;`
			`backup db sample tablespace (userspace1);`
			
			The timestamp for this backup image is:
			
			20040611174124
			
			`restore db sample tablespace (userspace1) online taken at 20040611174124;`
			
			While the restore operation is running, execute the following script from another session:
			
			`connect to sample;`
			
			1.
			
			`list tablespaces show detail;`
			
			or
			
			2.
			
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			Information returned for USERSPACE1 shows that this table space is in Restore in Progress state.

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 **logretain** database configuration parameter is set to RECOVERY, or the **userexit** database configuration parameter is set to YES. 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.	When the online table space restore operation in Restore in Progress completes, the table space USERSPACE1 is in Roll Forward Pending state.

Table 149. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Roll Forward in Progress	0x40	This is a transient state that is only in effect during a rollforward operation.	Given load input file `staffdata.del` with a substantial amount of data (for example, 20000 or more records):
			`update db cfg for sample using logretain recovery;`
			`backup db sample;`
			`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;`
			`connect reset;`
			`backup db sample tablespace (ts1) online;`
			
			The timestamp for this backup image is:
			
			20040630000715
			
			`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;`
			
			While the rollforward operation is running, execute the following script from another session:
			`connect to sample;`
			
			1.
			`list tablespaces show detail;`
			
			or
			
			2.
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			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, if you wish.	`backup db sample;`
			
			Assuming that the timestamp for this backup image is 20040613204955:
			`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 149. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

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.	`backup db sample;`
			
			Assuming that the timestamp for this backup image is 20040613204955:
			`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.

Table Space Creation in Progress	0x40000000	This is a transient state that is only in effect during a create table space operation.	`connect to sample;`
			`create tablespace ts1 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001/tsc1' 1024);`
			`create tablespace ts2 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001/tsc2' 1024);`
			`create tablespace ts3 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001/tsc3' 1024);`
			
			While the create table space operations are running, execute the following script from another session:
			`connect to sample;`
			
			1.
			`list tablespaces show detail;`
			
			or
			
			2.
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			Information returned for TS1, TS2, and TS3 shows that these table spaces are in Table Space Creation in Progress state.

Table 149. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Table Space Deletion in Progress	0x20000000	This is a transient state that is only in effect during a delete table space operation.	`connect to sample;`
			`create tablespace ts1 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001 /tsc1' 1024);`
			`create tablespace ts2 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001 /tsc2' 1024);`
			`create tablespace ts3 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001 /tsc3' 1024);`
			`drop tablespace ts1;`
			`drop tablespace ts2;`
			`drop tablespace ts3;`
			
			While the drop table space operations are running, execute the following script from another session:
			`connect to sample;`
			1.
			`list tablespaces show detail;`
			or
			2.
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			Information returned for TS1, TS2, and TS3 shows that these table spaces are in Table Space Deletion in Progress state.

LOAD

Loads data into a DB2 table. Data residing on the server can be in the form of a file, tape, or named pipe. If the COMPRESS attribute for the table is set to YES, the data loaded will be subject to compression on every data and database partition for which a dictionary already exists in the table, including data in the XML storage object of the table.

Quick link to "File type modifiers for the load utility" on page 622.

Restrictions

The load utility does not support loading data at the hierarchy level. The load utility is not compatible with range-clustered tables.

Scope

This command can be issued against multiple database partitions in a single request.

Authorization

One of the following:

- DATAACCESS
- LOAD authority on the database and
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.
- To load data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the load fails and an error (SQLSTATE 5U014) is returned.
- To load data into a table that has protected rows, the session authorization id must hold a security label that meets these criteria:
  - It is part of the security policy protecting the table
  - It was granted to the session authorization ID for write access or for all access

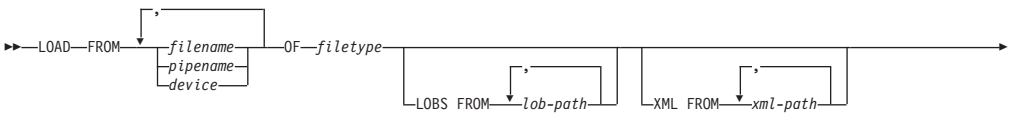
  If the session authorization id does not hold such a security label then the load fails and an error (SQLSTATE 5U014) is returned. This security label is used to protect a loaded row if the session authorization ID's LBAC credentials do not allow it to write to the security label that protects that row in the data. This does not happen, however, when the security policy protecting the table was created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option of the CREATE SECURITY POLICY statement. In this case the load fails and an error (SQLSTATE 42519) is returned.
- If the REPLACE option is specified, the session authorization ID must have the authority to drop the table.
- If the **LOCK WITH FORCE** option is specified, SYSADM authority is required.

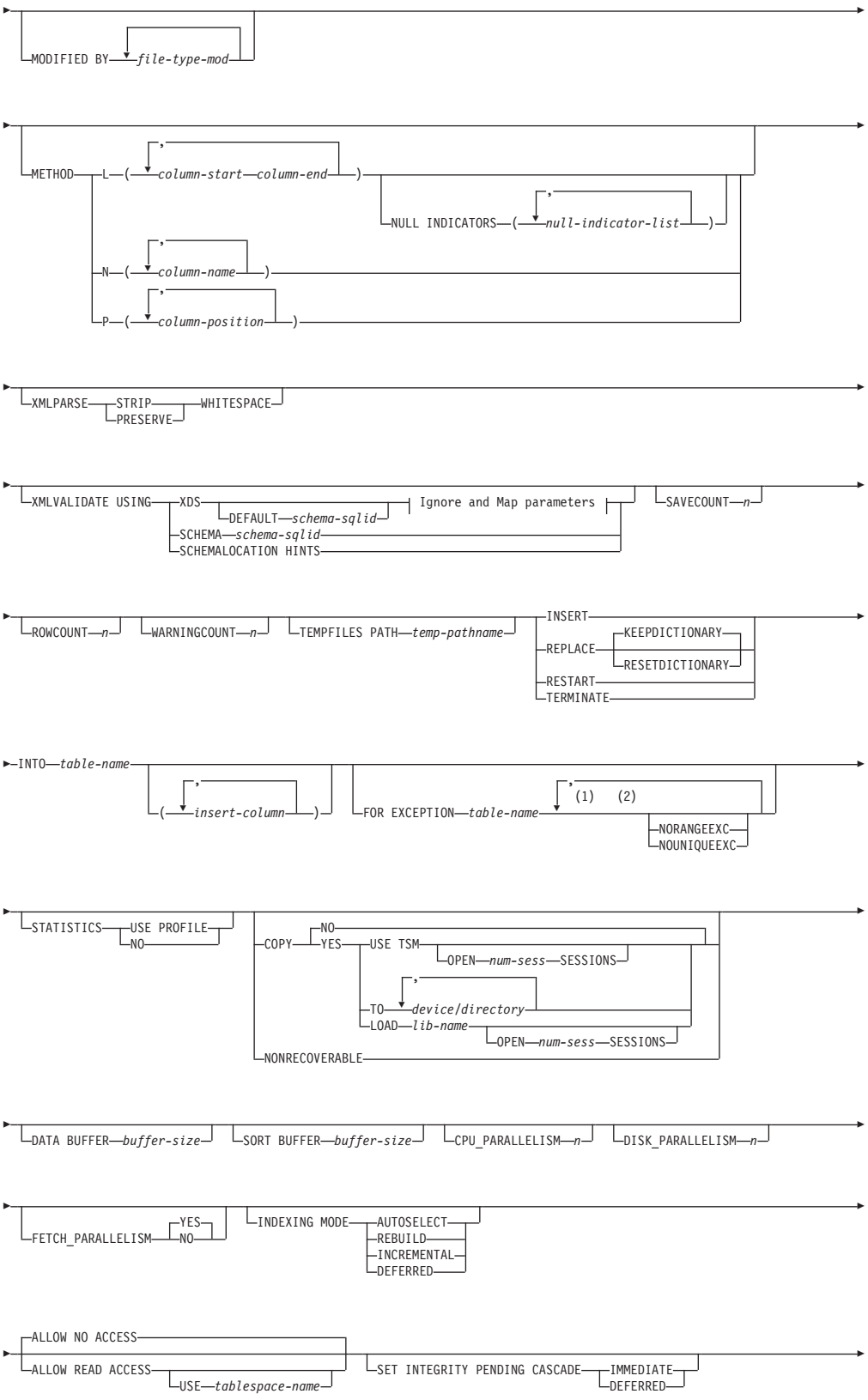
Since all load processes (and all DB2 server processes, in general) are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

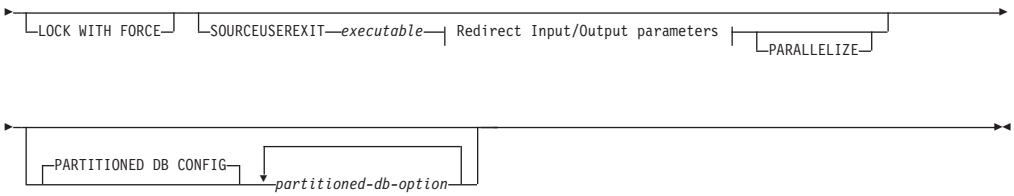
Required connection

Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

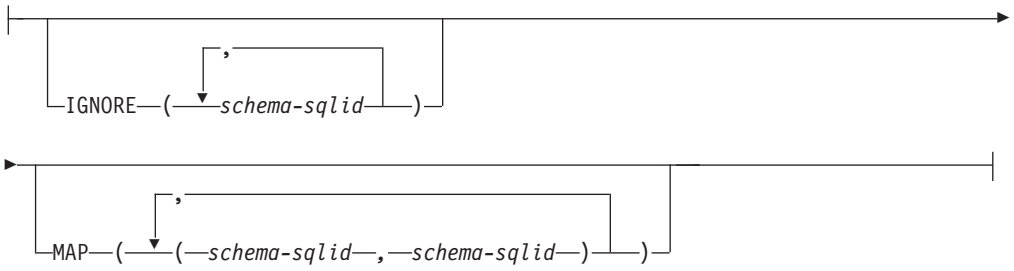
Command syntax



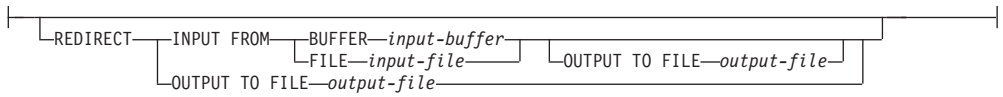

MODIFIED BY file-type-mod
METHOD
L (column-start column-end)
NULL INDICATORS (null-indicator-list)
N (column-name)
P (column-position)
XMLPARSE STRIP WHITESPACE
PRESERVE
XMLVALIDATE USING
XDS
DEFAULT schema-sqlid
SCHEMA schema-sqlid
SCHEMALOCATION HINTS
Ignore and Map parameters
SAVECOUNT n
ROWCOUNT n
WARNINGCOUNT n
TEMPFILES PATH temp-pathname
INSERT
REPLACE
KEEPDICTIONARY
RESETDICTIONARY
RESTART
TERMINATE
INTO table-name
(insert-column)
FOR EXCEPTION table-name (1) (2)
NORANGEEXC
NOUNIQUEEXC
STATISTICS
USE PROFILE
NO
COPY
NO
YES
USE TSM
OPEN num-sess SESSIONS
TO device/directory
LOAD lib-name
OPEN num-sess SESSIONS
NONRECOVERABLE
DATA BUFFER buffer-size
SORT BUFFER buffer-size
CPU_PARALLELISM n
DISK_PARALLELISM n
FETCH_PARALLELISM
YES
NO
INDEXING MODE
AUTOSELECT
REBUILD
INCREMENTAL
DEFERRED
ALLOW NO ACCESS
ALLOW READ ACCESS
USE tablespace-name
SET INTEGRITY PENDING CASCADE
IMMEDIATE
DEFERRED



Ignore and Map parameters:



Redirect Input/Output parameters:



Notes:

1 These keywords can appear in any order.

2 Each of these keywords can only appear once.

Command parameters

FROM *filename* | *pipename* | *device*

Note:

1. If data is exported into a file using the EXPORT command using the ADMIN_CMD procedure, the data file is owned by the fenced user ID. This file is not usually accessible by the instance owner. To run the LOAD from CLP or the ADMIN_CMD procedure, the data file must be accessible by the instance owner ID, so read access to the data file must be granted to the instance owner.
2. Loading data from multiple IXF files is supported if the files are physically separate, but logically one file. It is *not* supported if the files are both logically and physically separate. (Multiple physical files would be considered logically one if they were all created with one invocation of the EXPORT command.)
3. When loading XML data from files into tables in a partitioned database environment, the XML data files must be read-accessible to all the database partitions where loading is taking place.

OF *filetype*

Specifies the format of the data:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format)

- IXF (Integration Exchange Format, PC version) is a binary format that is used exclusively by DB2 databases.
- CURSOR (a cursor declared against a SELECT or VALUES statement).

Note: When using a CURSOR file type to load XML data into a table in a distributed database environment, the PARTITION_ONLY and LOAD_ONLY modes are not supported.

LOBS FROM *lob-path*
The path to the data files containing LOB values to be loaded. The path must end with a slash. The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. The maximum number of paths that can be specified is 999. This will implicitly activate the **LOBSINFILE** behavior.

This option is ignored when specified in conjunction with the CURSOR file type.

MODIFIED BY *file-type-mod*
Specifies file type modifier options. See "File type modifiers for the load utility" on page 622.

METHOD

L Specifies the start and end column numbers from which to load data. A column number is a byte offset from the beginning of a row of data. It is numbered starting from 1. This method can only be used with ASC files, and is the only valid method for that file type.

NULL INDICATORS *null-indicator-list*
This option can only be used when the **METHOD L** parameter is specified; that is, the input file is an ASC file). The null indicator list is a comma-separated list of positive integers specifying the column number of each null indicator field. The column number is the byte offset of the null indicator field from the beginning of a row of data. There must be one entry in the null indicator list for each data field defined in the **METHOD L** parameter. A column number of zero indicates that the corresponding data field always contains data.

A value of `Y` in the NULL indicator column specifies that the column data is NULL. Any character *other than* `Y` in the NULL indicator column specifies that the column data is not NULL, and that column data specified by the **METHOD L** option will be loaded.

The NULL indicator character can be changed using the **MODIFIED BY** option.

N Specifies the names of the columns in the data file to be loaded. The case of these column names must match the case of the corresponding names in the system catalogs. Each table column that is not nullable should have a corresponding entry in the **METHOD N** list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method N (F2, F1, F4, F3)` is a valid request, while `method N (F2, F1)` is not valid. This method can only be used with file types IXF or CURSOR.

P Specifies the field numbers (numbered from 1) of the input data fields to be loaded. Each table column that is not nullable should have a corresponding entry in the **METHOD P** list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method P (2, 1, 4, 3)` is a valid request, while `method P (2, 1)` is not valid. This method can only be used with file types IXF, DEL, or CURSOR, and is the only valid method for the DEL file type.

XML FROM *xml-path*

Specifies one or more paths that contain the XML files. XDSs are contained in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the XML column.

XMLPARSE

Specifies how XML documents are parsed. If this option is not specified, the parsing behavior for XML documents will be determined by the value of the CURRENT XMLPARSE OPTION special register.

STRIP WHITESPACE

Specifies to remove whitespace when the XML document is parsed.

PRESERVE WHITESPACE

Specifies not to remove whitespace when the XML document is parsed.

XMLVALIDATE

Specifies that XML documents are validated against a schema, when applicable.

USING XDS

XML documents are validated against the XML schema identified by the XML Data Specifier (XDS) in the main data file. By default, if the **XMLVALIDATE** option is invoked with the **USING XDS** clause, the schema used to perform validation will be determined by the SCH attribute of the XDS. If an SCH attribute is not present in the XDS, no schema validation will occur unless a default schema is specified by the **DEFAULT** clause.

The **DEFAULT**, **IGNORE**, and **MAP** clauses can be used to modify the schema determination behavior. These three optional clauses apply directly to the specifications of the XDS, and not to each other. For example, if a schema is selected because it is specified by the **DEFAULT** clause, it will not be ignored if also specified by the **IGNORE** clause. Similarly, if a schema is selected because it is specified as the first part of a pair in the **MAP** clause, it will not be re-mapped if also specified in the second part of another **MAP** clause pair.

USING SCHEMA *schema-sqlid*

XML documents are validated against the XML schema with the specified SQL identifier. In this case, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

USING SCHEMALOCATION HINTS

XML documents are validated against the schemas identified by XML schema location hints in the source XML documents. If a schemaLocation attribute is not found in the XML document, no validation will occur. When the **USING SCHEMALOCATION**

HINTS clause is specified, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

See examples of the **XMLVALIDATE** option below.

IGNORE *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. The **IGNORE** clause specifies a list of one or more schemas to ignore if they are identified by an SCH attribute. If an SCH attribute exists in the XML Data Specifier for a loaded XML document, and the schema identified by the SCH attribute is included in the list of schemas to ignore, then no schema validation will occur for the loaded XML document.

Note:

If a schema is specified in the **IGNORE** clause, it cannot also be present in the left side of a schema pair in the **MAP** clause.

The **IGNORE** clause applies only to the XDS. A schema that is mapped by the **MAP** clause will not be subsequently ignored if specified by the **IGNORE** clause.

DEFAULT *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. The schema specified through the **DEFAULT** clause identifies a schema to use for validation when the XML Data Specifier (XDS) of a loaded XML document does not contain an SCH attribute identifying an XML Schema.

The **DEFAULT** clause takes precedence over the **IGNORE** and **MAP** clauses. If an XDS satisfies the **DEFAULT** clause, the **IGNORE** and **MAP** specifications will be ignored.

MAP *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. Use the **MAP** clause to specify alternate schemas to use in place of those specified by the SCH attribute of an XML Data Specifier (XDS) for each loaded XML document. The **MAP** clause specifies a list of one or more schema pairs, where each pair represents a mapping of one schema to another. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

If a schema is present in the left side of a schema pair in the **MAP** clause, it cannot also be specified in the **IGNORE** clause.

Once a schema pair mapping is applied, the result is final. The mapping operation is non-transitive, and therefore the schema chosen will not be subsequently applied to another schema pair mapping.

A schema cannot be mapped more than once, meaning that it cannot appear on the left side of more than one pair.

SAVECOUNT *n*
Specifies that the load utility is to establish consistency points after every *n* rows. This value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using

LOAD QUERY. If the value of *n* is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance.

The default value is zero, meaning that no consistency points will be established, unless necessary.

This option is ignored when specified in conjunction with the CURSOR file type or when loading a table containing an XML column.

ROWCOUNT *n*

Specifies the number of *n* physical records in the file to be loaded. Allows a user to load only the first *n* rows in a file.

WARNINGCOUNT *n*

Stops the load operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the load file or the target table is specified incorrectly, the load utility will generate a warning for each row that it attempts to load, which will cause the load to fail. If *n* is zero, or this option is not specified, the load operation will continue regardless of the number of warnings issued. If the load operation is stopped because the threshold of warnings was encountered, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

TEMPFILES PATH *temp-pathname*

Specifies the name of the path to be used when creating temporary files during a load operation, and should be fully qualified according to the server database partition.

Temporary files take up file system space. Sometimes, this space requirement is quite substantial. Following is an estimate of how much file system space should be allocated for all temporary files:

- 136 bytes for each message that the load utility generates
- 15 KB overhead if the data file contains long field data or LOBs. This quantity can grow significantly if the **INSERT** option is specified, and there is a large amount of long field or LOB data already in the table.

INSERT

One of four modes under which the load utility can execute. Adds the loaded data to the table without changing the existing table data.

REPLACE

One of four modes under which the load utility can execute. Deletes all existing data from the table, and inserts the loaded data. The table definition and index definitions are not changed. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

KEEPDICTIONARY

An existing compression dictionary is preserved across the LOAD REPLACE operation. Provided the table COMPRESS attribute is YES, the newly replaced data is subject to being compressed using the dictionary that existed prior to the invocation of the load. If no dictionary previously existed in the table, a new dictionary is built using the data that is being replaced into the table as long as the table COMPRESS attribute is YES. The amount of data that is required to build the compression dictionary in this case is subject

to the policies of ADC. This data is populated into the table as uncompressed. Once the dictionary is inserted into the table, the remaining data to be loaded is subject to being compressed with this dictionary. This is the default parameter. For summary, see Table 1 below.

Table 150. LOAD REPLACE KEEPDICTIONARY

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Compression dictionary	Data compression
YES	YES	YES	Preserve table row data and XML dictionaries.	Data to be loaded is subject to compression.
YES	YES	NO	Preserve table row data dictionary and build a new XML dictionary.	Table row data to be loaded is subject to compression. After XML dictionary is built, remaining XML data to be loaded is subject to compression.
YES	NO	YES	Build table row data dictionary and preserve XML dictionary.	After table row data dictionary is built, remaining table row data to be loaded is subject to compression. XML data to be loaded is subject to compression.
YES	NO	NO	Build new table row data and XML dictionaries.	After dictionaries are built, remaining data to be loaded is subject to compression.
NO	YES	YES	Preserve table row data and XML dictionaries.	Data to be loaded is not compressed.
NO	YES	NO	Preserve table row data dictionary.	Data to be loaded is not compressed.
NO	NO	YES	No effect on table row dictionary. Preserve XML dictionary.	Data to be loaded is not compressed.
NO	NO	NO	No effect.	Data to be loaded is not compressed.

Note:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 Version 9.7 or later, or if the table is migrated using an online table move.

RESETDICTIONARY

This directive instructs LOAD REPLACE processing to build a new dictionary for the table data object provided that the table COMPRESS attribute is YES. If the COMPRESS attribute is NO and a dictionary was already present in the table it will be removed and no new dictionary will be inserted into the table. A compression dictionary can be built with just one user record. If the loaded data set size is zero and if there is a preexisting dictionary, the dictionary will not be preserved. The amount of data required to build a dictionary with this directive is not subject to the policies of ADC. For summary, see Table 2 below.

Table 151. LOAD REPLACE RESETDICTIONARY

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Compression dictionary	Data compression
YES	YES	YES	Build new dictionaries[2]. If the DATA CAPTURE CHANGES option is enabled on the CREATE TABLE or ALTER TABLE statements, the current table row data dictionary is kept (and referred to as the *historical compression dictionary*).	After dictionaries are built, remaining data to be loaded is subject to compression.
YES	YES	NO	Build new dictionaries[2]. If the DATA CAPTURE CHANGES option is enabled on the CREATE TABLE or ALTER TABLE statements, the current table row data dictionary is kept (and referred to as the *historical compression dictionary*).	After dictionaries are built, remaining data to be loaded is subject to compression.
YES	NO	YES	Build new dictionaries.	After dictionaries are built, remaining data to be loaded is subject to compression.
YES	NO	NO	Build new dictionaries.	After dictionaries are built, remaining data to be loaded is subject to compression.
NO	YES	YES	Remove dictionaries.	Data to be loaded is not compressed.
NO	YES	NO	Remove table row data dictionary.	Data to be loaded is not compressed.
NO	NO	YES	Remove XML storage object dictionary.	Data to be loaded is not compressed.
NO	NO	NO	No effect.	All table data is not compressed.

Notes:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 Version 9.7 or later, or if the table is migrated using an online table move.
2. If a dictionary exists and the compression attribute is enabled, but there are no records to load into the table partition, a new dictionary cannot be built and the **RESETDICTIONARY** operation will not keep the existing dictionary.

TERMINATE

One of four modes under which the load utility can execute. Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects might be marked as

invalid, in which case index rebuild will automatically take place at next access). If the load operation being terminated is a LOAD REPLACE, the table will be truncated to an empty table after the LOAD TERMINATE operation. If the load operation being terminated is a LOAD INSERT, the table will retain all of its original records after the LOAD TERMINATE operation. For summary of dictionary management, see Table 3 below.

The LOAD **TERMINATE** option will not remove a backup pending state from table spaces.

RESTART
One of four modes under which the load utility can execute. Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase. For summary of dictionary management, see Table 4 below.

INTO *table-name*
Specifies the database table into which the data is to be loaded. This table cannot be a system table, a declared temporary table, or a created temporary table. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form *schema.tablename*. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

insert-column
Specifies the table column into which the data is to be inserted.

The load utility cannot parse columns whose names contain one or more spaces. For example,

will fail because of the `Int 4` column. The solution is to enclose such column names with double quotation marks:

FOR EXCEPTION *table-name*
Specifies the exception table into which rows in error will be copied. Any row that is in violation of a unique index or a primary key index is copied. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

Information that is written to the exception table is *not* written to the dump file. In a partitioned database environment, an exception table must be defined for those database partitions on which the loading table is defined. The dump file, otherwise, contains rows that cannot be loaded because they are invalid or have syntax errors.

When loading XML data, using the **FOR EXCEPTION** clause to specify a load exception table is not supported in the following cases:

- When using label-based access control (LBAC).
- When loading data into a partitioned table.

NORANGEEXC
Indicates that if a row is rejected because of a range violation it will not be inserted into the exception table.

NOUNIQUEEXC
Indicates that if a row is rejected because it violates a unique constraint it will not be inserted into the exception table.

STATISTICS USE PROFILE
Instructs load to collect statistics during the load according to the profile defined for this table. This profile must be created before load is executed.

The profile is created by the RUNSTATS command. If the profile does not exist and load is instructed to collect statistics according to the profile, a warning is returned and no statistics are collected.

STATISTICS NO
Specifies that no statistics are to be collected, and that the statistics in the catalogs are not to be altered. This is the default.

COPY NO
Specifies that the table space in which the table resides will be placed in backup pending state if forward recovery is enabled (that is, **logretain** or **userexit** is on). The **COPY NO** option will also put the table space state into the Load in Progress table space state. This is a transient state that will disappear when the load completes or aborts. The data in any table in the table space cannot be updated or deleted until a table space backup or a full database backup is made. However, it is possible to access the data in any table by using the SELECT statement.

LOAD with **COPY NO** on a recoverable database leaves the table spaces in a backup pending state. For example, performing a LOAD with **COPY NO** and **INDEXING MODE DEFERRED** will leave indexes needing a refresh. Certain queries on the table might require an index scan and will not succeed until the indexes are refreshed. The index cannot be refreshed if it resides in a table space which is in the backup pending state. In that case, access to the table will not be allowed until a backup is taken. Index refresh is done automatically by the database when the index is accessed by a query. If one of **COPY NO**, **COPY YES**, or **NONRECOVERABLE** is not specified, and the database is recoverable (**logretain** or **logarchmeth1** is enabled), then **COPY NO** is the default.

COPY YES
Specifies that a copy of the loaded data will be saved. This option is invalid if forward recovery is disabled.

USE TSM
Specifies that the copy will be stored using Tivoli Storage Manager (TSM).

OPEN *num-sess* **SESSIONS**
The number of I/O sessions to be used with TSM or the vendor product. The default value is 1.

TO *device/directory*
Specifies the device or directory on which the copy image will be created.

LOAD *lib-name*
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. It can contain the full path. If the full path is not given, it will default to the path where the user exit programs reside.

NONRECOVERABLE
Specifies that the load transaction is to be marked as nonrecoverable and that it will not be possible to recover it by a subsequent roll forward action. The roll forward utility will skip the transaction and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the roll forward operation is completed, such a table can only be dropped or

restored from a backup (full or table space) taken after a commit point following the completion of the non-recoverable load operation.

With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation. If one of **COPY NO**, **COPY YES**, or **NONRECOVERABLE** is not specified, and the database is not recoverable (**logretain** or **logarchmeth1** is not enabled), then **NONRECOVERABLE** is the default.

WITHOUT PROMPTING
Specifies that the list of data files contains all the files that are to be loaded, and that the devices or directories listed are sufficient for the entire load operation. If a continuation input file is not found, or the copy targets are filled before the load operation finishes, the load operation will fail, and the table will remain in load pending state.

DATA BUFFER *buffer-size*
Specifies the number of 4 KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the minimum required resource is used, and no warning is returned.

This memory is allocated directly from the utility heap, whose size can be modified through the **util_heap_sz** database configuration parameter. Beginning in version 9.5, the value of the DATA BUFFER option of the LOAD command can temporarily exceed **util_heap_sz** if more memory is available in the system. In this situation, the utility heap is dynamically increased as needed until the **database_memory** limit is reached. This memory will be released once the load operation completes.

If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

SORT BUFFER *buffer-size*
This option specifies a value that overrides the **sortheap** database configuration parameter during a load operation. It is relevant only when loading tables with indexes and only when the **INDEXING MODE** parameter is not specified as DEFERRED. The value that is specified cannot exceed the value of **sortheap**. This parameter is useful for throttling the sort memory that is used when loading tables with many indexes without changing the value of **sortheap**, which would also affect general query processing.

CPU_PARALLELISM *n*
Specifies the number of processes or threads that the load utility will create for parsing, converting, and formatting records when building table objects. This parameter is designed to exploit the number of processes running per database partition. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, or has not been specified, the load utility uses an intelligent default value (usually based on the number of CPUs available) at run time.

Note:

1. If this parameter is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs or the value specified by the user.
2. Specifying a small value for the **SAVECOUNT** parameter causes the loader to perform many more I/O operations to flush both data and table metadata. When **CPU_PARALLELISM** is greater than one, the flushing operations are asynchronous, permitting the loader to exploit the CPU. When **CPU_PARALLELISM** is set to one, the loader waits on I/O during consistency points. A load operation with **CPU_PARALLELISM** set to two, and **SAVECOUNT** set to 10 000, completes faster than the same operation with **CPU_PARALLELISM** set to one, even though there is only one CPU.

DISK_PARALLELISM *n*
Specifies the number of processes or threads that the load utility will create for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

FETCH_PARALLELISM YES | NO
When performing a load from a cursor where the cursor is declared using the **DATABASE** keyword, or when using the API sqlu_remotefetch_entry media entry, and this option is set to YES, the load utility attempts to parallelize fetching from the remote data source if possible. If set to NO, no parallel fetching is performed. The default value is YES. For more information, see "Moving data using the CURSOR file type".

INDEXING MODE
Specifies whether the load utility is to rebuild indexes or to extend them incrementally. Valid values are:

AUTOSELECT
The load utility will automatically decide between REBUILD or INCREMENTAL mode. The decision is based on the amount of data being loaded and the depth of the index tree. Information relating to the depth of the index tree is stored in the index object. RUNSTATS is not required to populate this information. AUTOSELECT is the default indexing mode.

REBUILD
All indexes will be rebuilt. The utility must have sufficient resources to sort all index key parts for both old and appended table data.

INCREMENTAL
Indexes will be extended with new data. This approach consumes index free space. It only requires enough sort space to append index keys for the inserted records. This method is only supported in cases where the index object is valid and accessible at the start of a load operation (it is, for example, not valid immediately following a load operation in which the DEFERRED mode was specified). If this mode is specified, but not supported due to the state of the index, a warning is returned, and the load operation continues in REBUILD mode. Similarly, if a load restart operation is begun in the load build phase, INCREMENTAL mode is not supported.

DEFERRED
The load utility will not attempt index creation if this mode is

specified. Indexes will be marked as needing a refresh. The first access to such indexes that is unrelated to a load operation might force a rebuild, or indexes might be rebuilt when the database is restarted. This approach requires enough sort space for all key parts for the largest index. The total time subsequently taken for index construction is longer than that required in REBUILD mode. Therefore, when performing multiple load operations with deferred indexing, it is advisable (from a performance viewpoint) to let the last load operation in the sequence perform an index rebuild, rather than allow indexes to be rebuilt at first non-load access.

Deferred indexing is only supported for tables with non-unique indexes, so that duplicate keys inserted during the load phase are not persistent after the load operation.

ALLOW NO ACCESS
Load will lock the target table for exclusive access during the load. The table state will be set to Load In Progress during the load. **ALLOW NO ACCESS** is the default behavior. It is the only valid option for LOAD REPLACE.

When there are constraints on the table, the table state will be set to Set Integrity Pending as well as Load In Progress. The SET INTEGRITY statement must be used to take the table out of Set Integrity Pending state.

ALLOW READ ACCESS
Load will lock the target table in a share mode. The table state will be set to both Load In Progress and Read Access. Readers can access the non-delta portion of the data while the table is being load. In other words, data that existed before the start of the load will be accessible by readers to the table, data that is being loaded is not available until the load is complete. LOAD TERMINATE or LOAD RESTART of an **ALLOW READ ACCESS** load can use this option; LOAD TERMINATE or LOAD RESTART of an **ALLOW NO ACCESS** load cannot use this option. Furthermore, this option is not valid if the indexes on the target table are marked as requiring a rebuild.

When there are constraints on the table, the table state will be set to Set Integrity Pending as well as Load In Progress, and Read Access. At the end of the load, the table state Load In Progress will be removed but the table states Set Integrity Pending and Read Access will remain. The SET INTEGRITY statement must be used to take the table out of Set Integrity Pending. While the table is in Set Integrity Pending and Read Access states, the non-delta portion of the data is still accessible to readers, the new (delta) portion of the data will remain inaccessible until the SET INTEGRITY statement has completed. A user can perform multiple loads on the same table without issuing a SET INTEGRITY statement. Only the original (checked) data will remain visible, however, until the SET INTEGRITY statement is issued.

ALLOW READ ACCESS also supports the following modifiers:

USE *tablespace-name*
If the indexes are being rebuilt, a shadow copy of the index is built in table space *tablespace-name* and copied over to the original table space at the end of the load during an INDEX COPY PHASE. Only system temporary table spaces can be used with this option. If not specified then the shadow index will be created in the same table space as the index object. If the shadow copy is created in the same

table space as the index object, the copy of the shadow index object over the old index object is instantaneous. If the shadow copy is in a different table space from the index object a physical copy is performed. This could involve considerable I/O and time. The copy happens while the table is offline at the end of a load during the INDEX COPY PHASE.

Without this option the shadow index is built in the same table space as the original. Since both the original index and shadow index by default reside in the same table space simultaneously, there might be insufficient space to hold both indexes within one table space. Using this option ensures that you retain enough table space for the indexes.

This option is ignored if the user does not specify **INDEXING MODE REBUILD** or **INDEXING MODE AUTOSELECT**. This option will also be ignored if **INDEXING MODE AUTOSELECT** is chosen and load chooses to incrementally update the index.

SET INTEGRITY PENDING CASCADE

If LOAD puts the table into Set Integrity Pending state, the **SET INTEGRITY PENDING CASCADE** option allows the user to specify whether or not Set Integrity Pending state of the loaded table is immediately cascaded to all descendents (including descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables).

IMMEDIATE

Indicates that Set Integrity Pending state is immediately extended to all descendent foreign key tables, descendent immediate materialized query tables and descendent staging tables. For a LOAD INSERT operation, Set Integrity Pending state is not extended to descendent foreign key tables even if the **IMMEDIATE** option is specified.

When the loaded table is later checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statement), descendent foreign key tables that were placed in Set Integrity Pending Read Access state will be put into Set Integrity Pending No Access state.

DEFERRED

Indicates that only the loaded table will be placed in the Set Integrity Pending state. The states of the descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged.

Descendent foreign key tables might later be implicitly placed in Set Integrity Pending state when their parent tables are checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statement). Descendent immediate materialized query tables and descendent immediate staging tables will be implicitly placed in Set Integrity Pending state when one of its underlying tables is checked for integrity violations. A warning (SQLSTATE 01586) will be issued to indicate that dependent tables have been placed in Set Integrity Pending state. See the Notes section of the SET INTEGRITY statement in the SQL Reference for when these descendent tables will be put into Set Integrity Pending state.

If the SET INTEGRITY PENDING CASCADE option is not specified:

- Only the loaded table will be placed in Set Integrity Pending state. The state of descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged, and can later be implicitly put into Set Integrity Pending state when the loaded table is checked for constraint violations.

If LOAD does not put the target table into Set Integrity Pending state, the **SET INTEGRITY PENDING CASCADE** option is ignored.

LOCK WITH FORCE
The utility acquires various locks including table locks in the process of loading. Rather than wait, and possibly timeout, when acquiring a lock, this option allows load to force off other applications that hold conflicting locks on the target table. Applications holding conflicting locks on the system catalog tables will not be forced off by the load utility. Forced applications will roll back and release the locks the load utility needs. The load utility can then proceed. This option requires the same authority as the FORCE APPLICATIONS command (SYSADM or SYSCTRL).

ALLOW NO ACCESS loads might force applications holding conflicting locks at the start of the load operation. At the start of the load the utility can force applications that are attempting to either query or modify the table.

ALLOW READ ACCESS loads can force applications holding conflicting locks at the start or end of the load operation. At the start of the load the load utility can force applications that are attempting to modify the table. At the end of the load operation, the load utility can force applications that are attempting to either query or modify the table.

SOURCEUSEREXIT *executable*
Specifies an executable filename which will be called to feed data into the utility.

REDIRECT

INPUT FROM

BUFFER *input-buffer*
The stream of bytes specified in *input-buffer* is passed into the STDIN file descriptor of the process executing the given executable.

FILE *input-file*
The contents of this client-side file are passed into the STDIN file descriptor of the process executing the given executable.

OUTPUT TO

FILE *output-file*
The STDOUT and STDERR file descriptors are captured to the fully qualified server-side file specified.

PARALLELIZE
Increases the throughput of data coming into the load utility by invoking multiple user exit processes simultaneously. This option is only applicable in multi-partition database environments and is ignored in single-partition database environments.

For more information, see "Moving data using a customized application (user exit)".

PARTITIONED DB CONFIG *partitioned-db-option*
Allows you to execute a load into a table distributed across multiple database partitions. The **PARTITIONED DB CONFIG** parameter allows you to specify partitioned database-specific configuration options. The *partitioned-db-option* values can be any of the following:

```
PART_FILE_LOCATION x
OUTPUT_DBPARTNUMS x
PARTITIONING_DBPARTNUMS x
MODE x
MAX_NUM_PART_AGENTS x
ISOLATE_PART_ERRS x
STATUS_INTERVAL x
PORT_RANGE x
CHECK_TRUNCATION
MAP_FILE_INPUT x
MAP_FILE_OUTPUT x
TRACE x
NEWLINE
DISTFILE x
OMIT_HEADER
RUN_STAT_DBPARTNUM x
```

Detailed descriptions of these options are provided in "Load configuration options for partitioned database environments".

RESTARTCOUNT
Reserved.

USING *directory*
Reserved.

Examples of loading data from XML documents

Loading XML data

Example 1

The user has constructed a data file with XDS fields to describe the documents that are to be inserted into the table. It might appear like this :

```
1, "<XDS FIL=""file1.xml"" />"
2, "<XDS FIL='file2.xml' OFF='23' LEN='45' />"
```

For the first row, the XML document is identified by the file named `file1.xml`. Note that since the character delimiter is the double quote character, and double quotation marks exist inside the XDS, the double quotation marks contained within the XDS are doubled. For the second row, the XML document is identified by the file named `file2.xml`, and starts at byte offset 23, and is 45 bytes in length.

Example 2

The user issues a load command without any parsing or validation options for the XML column, and the data is loaded successfully:

```
LOAD
FROM data.del of DEL INSERT INTO mytable
```

Loading XML data from CURSOR

Loading data from cursor is the same as with a regular relational column type. The user has two tables, T1 and T2, each of which consist of a single XML column named C1. To LOAD from T1 into T2, the user will first declare a cursor:

```
DECLARE
X1 CURSOR FOR SELECT C1 FROM T1;
```

Next, the user may issue a LOAD using the cursor type:

```
LOAD FROM X1 of
CURSOR INSERT INTO T2
```

Applying the XML specific LOAD options to the cursor type is the same as loading from a file.

Usage notes

- Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a load is attempted. If preservation of the source data order is not required, consider using the **ANYORDER** file type modifier, described below in the "File type modifiers for the load utility" section.
- The load utility builds indexes based on existing definitions. The exception tables are used to handle duplicates on unique keys. The utility does not enforce referential integrity, perform constraints checking, or update materialized query tables that are dependent on the tables being loaded. Tables that include referential or check constraints are placed in Set Integrity Pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in Set Integrity Pending state. Issue the SET INTEGRITY statement to take the tables out of Set Integrity Pending state. Load operations cannot be carried out on replicated materialized query tables.
- If a clustering index exists on the table, the data should be sorted on the clustering index prior to loading. Data does not need to be sorted prior to loading into a multidimensional clustering (MDC) table, however.
- If you specify an exception table when loading into a protected table, any rows that are protected by invalid security labels will be sent to that table. This might allow users that have access to the exception table to access to data that they would not normally be authorized to access. For better security be careful who you grant exception table access to, delete each row as soon as it is repaired and copied to the table being loaded, and drop the exception table as soon as you are done with it.
- Security labels in their internal format might contain newline characters. If you load the file using the DEL file format, those newline characters can be mistaken for delimiters. If you have this problem use the older default priority for delimiters by specifying the **delprioritychar** file type modifier in the LOAD command.
- For performing a load using the CURSOR file type where the DATABASE keyword was specified during the DECLARE CURSOR statement, the user ID and password used to authenticate against the database currently connected to (for the load) will be used to authenticate against the source database (specified by the DATABASE option of the DECLARE CURSOR statement). If no user ID or password was specified for the connection to the loading database, a user ID and password for the source database must be specified during the DECLARE CURSOR statement.
- Loading a multiple-part PC/IXF file whose individual parts are copied from a Windows system to an AIX system is supported. The names of all the files must

be specified in the LOAD command. For example, `LOAD FROM DATA.IXF, DATA.002 OF IXF INSERT INTO TABLE1`. Loading to the Windows operating system from logically split PC/IXF files is not supported.

- When restarting a failed LOAD, the behavior will follow the existing behavior in that the BUILD phase will be forced to use the REBUILD mode for indexes.
- Loading XML documents between databases is not supported and returns error message SQL1407N.

Summary of LOAD TERMINATE and LOAD RESTART dictionary management

The following chart summarizes the compression dictionary management behavior for LOAD processing under the **TERMINATE** directive.

Table 152. LOAD TERMINATE dictionary management

Table COMPRESS attribute	Does table row data dictionary exist prior to LOAD?	XML storage object dictionary exists prior to LOAD[1]	TERMINATE: LOAD REPLACE KEEPDICTIONARY or LOAD INSERT	TERMINATE: LOAD REPLACE RESETDICTIONARY
YES	YES	YES	Keep existing dictionaries.	Neither dictionary is kept. [2]
YES	YES	NO	Keep existing dictionary.	Nothing is kept. [2]
YES	NO	YES	Keep existing dictionary.	Nothing is kept.
YES	NO	NO	Nothing is kept.	Nothing is kept.
NO	YES	YES	Keep existing dictionaries.	Nothing is kept.
NO	YES	NO	Keep existing dictionary.	Nothing is kept.
NO	NO	YES	Keep existing dictionary.	Nothing is kept.
NO	NO	NO	Do nothing.	Do nothing.

Note:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 Version 9.7 or later, or if the table is migrated using an online table move.
2. In the special case that the table has data capture enabled, the table row data dictionary is kept.

LOAD RESTART truncates a table up to the last consistency point reached. As part of LOAD RESTART processing, a compression dictionary will exist in the table if it was present in the table at the time the last LOAD consistency point was taken. In that case, LOAD RESTART will not create a new dictionary. For a summary of the possible conditions, see Table 4 below.

Table 153. LOAD RESTART dictionary management

Table COMPRESS Attribute	Table row data dictionary exist prior to LOAD consistency point?[1]	XML Storage object dictionary existed prior to last LOAD?[2]	RESTART: LOAD REPLACE KEEPDICTIONARY or LOAD INSERT	RESTART: LOAD REPLACE RESETDICTIONARY
YES	YES	YES	Keep existing dictionaries.	Keep existing dictionaries.

Table 153. LOAD RESTART dictionary management (continued)

Table COMPRESS Attribute	Table row data dictionary exist prior to LOAD consistency point?[1]	XML Storage object dictionary existed prior to last LOAD?[2]	RESTART: LOAD REPLACE KEEPDICTIONARY or LOAD INSERT	RESTART: LOAD REPLACE RESETDICTIONARY
YES	YES	NO	Keep existing table row data dictionary and build XML dictionary subject to ADC.	Keep existing table row data dictionary and build XML dictionary.
YES	NO	YES	Build table row data dictionary subject to ADC. Keep existing XML dictionary.	Build table row data dictionary. Keep existing XML dictionary.
YES	NO	NO	Build table row data and XML dictionaries subject to ADC.	Build table row data and XML dictionaries.
NO	YES	YES	Keep existing dictionaries.	Remove existing dictionaries.
NO	YES	NO	Keep existing table row data dictionary.	Remove existing table row data dictionary.
NO	NO	YES	Keep existing XML dictionary.	Remove existing XML dictionary.
NO	NO	NO	Do nothing.	Do nothing.

Notes:

1. The **SAVECOUNT** option is ignored when loading XML data, load operations that fail during the load phase restart from the beginning of the operation.
2. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 Version 9.7 or later, or if the table is migrated using an online table move.

File type modifiers for the load utility

Table 154. Valid file type modifiers for the load utility: All file formats

Modifier	Description
anyorder	This modifier is used in conjunction with the **cpu_parallelism** parameter. Specifies that the preservation of source data order is not required, yielding significant additional performance benefit on SMP systems. If the value of **cpu_parallelism** is 1, this option is ignored. This option is not supported if **SAVECOUNT** > 0, since crash recovery after a consistency point requires that data be loaded in sequence.
generatedignore	This modifier informs the load utility that data for all generated columns is present in the data file but should be ignored. This results in all generated column values being generated by the utility. This modifier cannot be used with either the **generatedmissing** or the **generatedoverride** modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated column (not even NULLs). This results in all generated column values being generated by the utility. This modifier cannot be used with either the **generatedignore** or the **generatedoverride** modifier.

Table 154. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description

generatedoverride	This modifier instructs the load utility to accept user-supplied data for all generated columns in the table (contrary to the normal rules for these types of columns). This is useful when migrating data from another database system, or when loading a table from data that was recovered using the **RECOVER DROPPED TABLE** option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for a non-nullable generated column will be rejected (SQL3116W). When this modifier is used, the table will be placed in Set Integrity Pending state. To take the table out of Set Integrity Pending state without verifying the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR table-name GENERATED COLUMN IMMEDIATE UNCHECKED`
	
	To take the table out of Set Integrity Pending state and force verification of the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR table-name IMMEDIATE CHECKED.`
	
	When this modifier is specified and there is a generated column in any of the partitioning keys, dimension keys or distribution keys, then the LOAD command will automatically convert the modifier to **generatedignore** and proceed with the load. This will have the effect of regenerating all of the generated column values.
	
	This modifier cannot be used with either the **generatedmissing** or the **generatedignore** modifier.

identityignore	This modifier informs the load utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with either the **identitymissing** or the **identityoverride** modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with either the **identityignore** or the **identityoverride** modifier.
identityoverride	This modifier should be used only when an identity column defined as GENERATED ALWAYS is present in the table to be loaded. It instructs the utility to accept explicit, non-NULL data for such a column (contrary to the normal rules for these types of identity columns). This is useful when migrating data from another database system when the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the **DROPPED TABLE RECOVERY** option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for the identity column will be rejected (SQL3116W). This modifier cannot be used with either the **identitymissing** or the **identityignore** modifier. The load utility will not attempt to maintain or verify the uniqueness of values in the table's identity column when this option is used.
indexfreespace=*x*	*x* is an integer between 0 and 99 inclusive. The value is interpreted as the percentage of each index page that is to be left as free space when load rebuilds the index. Load with **INDEXING MODE INCREMENTAL** ignores this option. The first entry in a page is added without restriction; subsequent entries are added to maintain the percent free space threshold. The default value is the one used at CREATE INDEX time. This value takes precedence over the PCTFREE value specified in the CREATE INDEX statement. The **indexfreespace** option affects index leaf pages only.

Table 154. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB data. The ASC, DEL, or IXF load input files contain the names of the files having LOB data in the LOB column. This option is not supported in conjunction with the CURSOR filetype. The **LOBS FROM** clause specifies where the LOB files are located when the **lobsinfile** modifier is used. The **LOBS FROM** clause will implicitly activate the **lobsinfile** behavior. The **LOBS FROM** clause conveys to the LOAD utility the list of paths to search for the LOB files while loading the data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename*.ext.*nnn*.*mmm*/, where *filename*.ext is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string db2exp.001.123.456/ is stored in the data file, the LOB is located at offset 123 in the file db2exp.001, and is 456 bytes long. To indicate a null LOB , enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
noheader	Skips the header verification code (applicable only to load operations into tables that reside in a single-partition database partition group). If the default MPP load (mode PARTITION_AND_LOAD) is used against a table residing in a single-partition database partition group, the file is not expected to have a header. Thus the **noheader** modifier is not needed. If the LOAD_ONLY mode is used, the file is expected to have a header. The only circumstance in which you should need to use the **noheader** modifier is if you wanted to perform LOAD_ONLY operation using a file that does not have a header.
norowwarnings	Suppresses all warnings about rejected rows.
pagefreespace=*x*	*x* is an integer between 0 and 100 inclusive. The value is interpreted as the percentage of each data page that is to be left as free space. If the specified value is invalid because of the minimum row size, (for example, a row that is at least 3 000 bytes long, and an *x* value of 50), the row will be placed on a new page. If a value of 100 is specified, each row will reside on a new page. The PCTFREE value of a table determines the amount of free space designated per page. If a **pagefreespace** value on the load operation or a PCTFREE value on a table have not been set, the utility will fill up as much space as possible on each page. The value set by **pagefreespace** overrides the PCTFREE value specified for the table.
rowchangetimestampignore	This modifier informs the load utility that data for the row change timestamp column is present in the data file but should be ignored. This results in all ROW CHANGE TIMESTAMPs being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with either the **rowchangetimestampmissing** or the **rowchangetimestampoverride** modifier.
rowchangetimestampmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the row change timestamp column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This modifier cannot be used with either the **rowchangetimestampignore** or the **rowchangetimestampoverride** modifier.

Table 154. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
rowchangetimestampoverride	This modifier should be used only when a row change timestamp column defined as GENERATED ALWAYS is present in the table to be loaded. It instructs the utility to accept explicit, non-NULL data for such a column (contrary to the normal rules for these types of row change timestamp columns). This is useful when migrating data from another database system when the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the **DROPPED TABLE RECOVERY** option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for the ROW CHANGE TIMESTAMP column will be rejected (SQL3116W). This modifier cannot be used with either the **rowchangetimestampmissing** or the **rowchangetimestampignore** modifier. The load utility will not attempt to maintain or verify the uniqueness of values in the table's row change timestamp column when this option is used.

seclabelchar	Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. LOAD converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3242W) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W) is returned.
	
	This modifier cannot be specified if the **seclabelname** modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.
	
	If you have a table consisting of a single `DB2SECURITYLABEL` column, the data file might look like this:
	`"CONFIDENTIAL:ALPHA:G2"`
	`"CONFIDENTIAL;SIGMA:G2"`
	`"TOP SECRET:ALPHA:G2"`
	
	To load or import this data, the **seclabelchar** file type modifier must be used:
	`LOAD FROM input.del OF DEL MODIFIED BY SECLABELCHAR INSERT INTO t1`

seclabelname	Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. LOAD will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned.
	
	This modifier cannot be specified if the **seclabelchar** modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.
	
	If you have a table consisting of a single `DB2SECURITYLABEL` column, the data file might consist of security label names similar to:
	`"LABEL1"`
	`"LABEL1"`
	`"LABEL2"`
	
	To load or import this data, the **seclabelname** file type modifier must be used:
	`LOAD FROM input.del OF DEL MODIFIED BY SECLABELNAME INSERT INTO t1`
	
	Note: If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the **striptblanks** file type modifier to make sure the spaces are removed.

Table 154. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
totalfreespace=*x*	*x* is an integer greater than or equal to 0. The value is interpreted as the percentage of the total pages in the table that is to be appended to the end of the table as free space. For example, if *x* is 20, and the table has 100 data pages after the data has been loaded, 20 additional empty pages will be appended. The total number of data pages for the table will be 120. The data pages total does not factor in the number of index pages in the table. This option does not affect the index object. If two loads are done with this option specified, the second load will not reuse the extra space appended to the end by the first load.
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (", ,") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL. Without this option, if a source column contains no data for a row instance, one of the following occurs: • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 155. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data (and numeric data specified in characters) from this code page to the database code page during the load operation. The following rules apply: • For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • For DEL data specified in an EBCDIC code page, the delimiters might not coincide with the shift-in and shift-out DBCS characters. • **nullindchar** must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. EBCDIC data can use the corresponding symbols, even though the code points will be different. This option is not supported in conjunction with the CURSOR filetype.

Table 155. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

dateformat=*"x"*	*x* is the format of the date in the source file.[1] Valid date elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 1 - 12; mutually exclusive with M)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

dumpfile = *x*	*x* is the fully qualified (according to the server database partition) name of an exception file to which rejected rows are written. A maximum of 32 KB of data is written per record. Following is an example that shows how to specify a dump file: `db2 load from data of del modified by dumpfile = /u/user/filename insert into table_name` The file will be created and owned by the instance owner. To override the default file permissions, use the **dumpfileaccessall** file type modifier. **Note:** 1. In a partitioned database environment, the path should be local to the loading database partition, so that concurrently running load operations do not attempt to write to the same file. 2. The contents of the file are written to disk in an asynchronous buffered mode. In the event of a failed or an interrupted load operation, the number of records committed to disk cannot be known with certainty, and consistency cannot be guaranteed after a LOAD RESTART. The file can only be assumed to be complete for a load operation that starts and completes in a single pass. 3. If the specified file already exists, it will not be recreated, but it will be appended.
dumpfileaccessall	Grants read access to 'OTHERS' when a dump file is created. This file type modifier is only valid when: 1. it is used in conjunction with **dumpfile** file type modifier 2. the user has SELECT privilege on the load target table 3. it is issued on a DB2 server database partition that resides on a UNIX operating system If the specified file already exists, its permissions will not be changed.
fastparse	Use with caution. Reduces syntax checking on user-supplied column values, and enhances performance. Tables are guaranteed to be architecturally correct (the utility performs sufficient data checking to prevent a segmentation violation or trap), however, the coherence of the data is not validated. Only use this option if you are certain that your data is coherent and correct. For example, if the user-supplied data contains an invalid timestamp column value of `:1>0-00-20-07.11.12.000000`, this value is inserted into the table if **fastparse** is specified, and rejected if **fastparse** is not specified.

Table 155. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`. This modifier cannot be used with the **packeddecimal** modifier.

timeformat="*x*"	*x* is the format of the time in the source file.[1] Valid time elements are:
	`H     - Hour (one or two digits ranging from 0 - 12`
	`          for a 12 hour system, and 0 - 24`
	`          for a 24 hour system)`
	`HH    - Hour (two digits ranging from 0 - 12`
	`          for a 12 hour system, and 0 - 24`
	`          for a 24 hour system; mutually exclusive`
	`            with H)`
	`M     - Minute (one or two digits ranging`
	`          from 0 - 59)`
	`MM    - Minute (two digits ranging from 0 - 59;`
	`          mutually exclusive with M)`
	`S     - Second (one or two digits ranging`
	`          from 0 - 59)`
	`SS    - Second (two digits ranging from 0 - 59;`
	`          mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits`
	`          ranging from 00000 - 86399; mutually`
	`          exclusive with other time elements)`
	`TT    - Meridian indicator (AM or PM)`
	
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	`"HH:MM:SS"`
	`"HH.MM TT"`
	`"SSSSS"`

Table 155. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[1] Valid time stamp elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)`
	`MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`U (1 to 12 times) - Fractional seconds(number of occurrences of U represent the number of digits with each digit ranging from 0 to 9`
	`TT - Meridian indicator (AM or PM)`

timestampformat="*x*" (Continued)	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 'Jan' is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The valid values for the MMM element include: 'jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov' and 'dec'. These values are case insensitive.
	If the **timestampformat** modifier is not specified, the load utility formats the timestamp field using one of two possible formats:
	`YYYY-MM-DD-HH.MM.SS`
	`YYYY-MM-DD HH:MM:SS`
	The load utility chooses the format by looking at the separator between the DD and HH. If it is a dash '-', the load utility uses the regular dashes and dots format (YYYY-MM-DD-HH.MM.SS). If it is a blank space, then the load utility expects a colon ':' to separate the HH, MM and SS.
	In either format, if you include the microseconds field (UUUUUU), the load utility expects the dot '.' as the separator. Either YYYY-MM-DD-HH.MM.SS.UUUUUU or YYYY-MM-DD HH:MM:SS.UUUUUU are acceptable.
	The following example illustrates how to load data containing user defined date and time formats into a table called schedule:
	`db2 load from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

Table 155. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
usegraphiccodepage	If **usegraphiccodepage** is given, the assumption is made that data being loaded into graphic or double-byte character large object (DBCLOB) data field(s) is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic codepage is associated with the character code page. LOAD determines the character code page through either the **codepage** modifier, if it is specified, or through the code page of the database if the **codepage** modifier is not specified. This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data. **Restrictions** The **usegraphiccodepage** modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The **usegraphiccodepage** modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.
xmlchar	Specifies that XML documents are encoded in the character code page. This option is useful for processing XML documents that are encoded in the specified character code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the character code page, otherwise the row containing the document will be rejected. Note that the character codepage is the value specified by the **codepage** file type modifier, or the application codepage if it is not specified. By default, either the documents are encoded in Unicode, or they contain a declaration tag with an encoding attribute.
xmlgraphic	Specifies that XML documents are encoded in the specified graphic code page. This option is useful for processing XML documents that are encoded in a specific graphic code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the graphic code page, otherwise the row containing the document will be rejected. Note that the graphic code page is the graphic component of the value specified by the **codepage** file type modifier, or the graphic component of the application code page if it is not specified. By default, documents are either encoded in Unicode, or they contain a declaration tag with an encoding attribute.

Table 156. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII)

Modifier	Description
binarynumerics	Numeric (but not DECIMAL) data must be in binary form, not the character representation. This avoids costly conversions. This option is supported only with positional ASC, using fixed length records specified by the **reclen** option. The following rules apply: • No conversion between data types is performed, with the exception of BIGINT, INTEGER, and SMALLINT. • Data lengths must match their target column definitions. • FLOATs must be in IEEE Floating Point format. • Binary data in the load source file is assumed to be big-endian, regardless of the platform on which the load operation is running. NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
nochecklengths	If **nochecklengths** is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
nullindchar=*x*	*x* is a single character. Changes the character denoting a NULL value to *x*. The default value of *x* is `Y`.[2] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the NULL indicator character is specified to be the letter `N`, then `n` is also recognized as a NULL indicator.

packeddecimal	Loads packed-decimal data directly, since the **binarynumerics** modifier does not include the DECIMAL field type.
	
	This option is supported only with positional ASC, using fixed length records specified by the **reclen** option.
	
	Supported values for the sign nibble are:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
	
	Regardless of the server platform, the byte order of binary data in the load source file is assumed to be big-endian; that is, when using this modifier on Windows operating systems, the byte order must not be reversed.
	
	This modifier cannot be used with the **implieddecimal** modifier.

reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a newline character is not used to indicate the end of the row.

Table 156. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII) (continued)

Modifier	Description
striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept. This option cannot be specified together with **striptnulls**. These are mutually exclusive options. This option replaces the obsolete **t** option, which is supported for earlier compatibility only.
striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with **striptblanks**. These are mutually exclusive options. This option replaces the obsolete **padwithzero** option, which is supported for earlier compatibility only.

zoneddecimal	Loads zoned decimal data, since the **binarynumerics** modifier does not include the DECIMAL field type. This option is supported only with positional ASC, using fixed length records specified by the **reclen** option.
	
	Half-byte sign values can be one of the following:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	Supported values for digits are `0x0` to `0x9`.
	
	Supported values for zones are `0x3` and `0xF`.

Table 157. Valid file type modifiers for the load utility: DEL file formats (Delimited ASCII)

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark (″). The specified character is used in place of double quotation marks to enclose a character string.[23] If you want to explicitly specify the double quotation mark (″) as the character string delimiter, you should specify it as follows:
	`modified by chardel""`
	
	The single quotation mark (′) can also be specified as a character string delimiter as follows:
	`modified by chardel''`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[23]
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[23]

Table 157. Valid file type modifiers for the load utility: DEL file formats (Delimited ASCII) (continued)

Modifier	Description

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 load ... modified by delprioritychar ...`
	
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	
	With the **delprioritychar** modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and trailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
	
	The following example illustrates how to load data into a table called TABLE1, while preserving all leading and trailing spaces in the data file:
	`db2 load from delfile3 of del`
	`  modified by keepblanks`
	`  insert into table1`

nochardel	The load utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using a DB2 database system (unless **nochardel** was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with **chardelx**, **delprioritychar** or **nodoubledel**. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.

Table 158. Valid file type modifiers for the load utility: IXF file format

Modifier	Description
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If **nochecklengths** is specified, no checking is done, and an attempt is made to load each row.
nochecklengths	If **nochecklengths** is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Note:

1. Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

   For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```

   Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
2. Character values provided for the **chardel**, **coldel**, or **decpt** file type modifiers must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```
3. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.
4. The load utility does not issue a warning if an attempt is made to use unsupported file types with the **MODIFIED BY** option. If this is attempted, the load operation fails, and an error code is returned.
5. When importing into a table containing an implicitly hidden row change timestamp column, the implicitly hidden property of the column is not honoured. Therefore, the **rowchangetimestampmissing** file type modifier *must be* specified in the IMPORT command if data for the column is not present in the data to be imported and there is no explicit column list present.

Table 159. LOAD behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	LOAD behavior
Absent	Absent	All data in the file is assumed to be in the database code page, not the application code page, even if the **CLIENT** option is specified.
Present	Absent	All data in the file is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when loaded into the database if `N` is a single-byte code page.

Table 159. LOAD behavior when using codepage and usegraphiccodepage (continued)

codepage=N	usegraphiccodepage	LOAD behavior
Absent	Present	Character data in the file is assumed to be in the database code page, even if the **CLIENT** option is specified. Graphic data is assumed to be in the code page of the database graphic data, even if the **CLIENT** option is specified. If the database code page is single-byte, then all data is assumed to be in the database code page. **Warning:** Graphic data will be corrupted when loaded into a single-byte database.
Present	Present	Character data is assumed to be in code page `N`. Graphic data is assumed to be in the graphic code page of `N`. If `N` is a single-byte or double-byte code page, then all data is assumed to be in code page `N`. **Warning:**Graphic data will be corrupted when loaded into the database if `N` is a single-byte code page.

UPGRADE DATABASE

Converts a DB2 database of the previous version to the formats corresponding to the release run by the instance.

The db2ckupgrade command must be issued prior to upgrading the instance to verify that your databases are ready for upgrade. The db2iupgrade command implicitly calls the db2ckupgrade. Backup all databases prior to upgrade, and prior to the installation of the current version of DB2 database product on Windows operating systems.

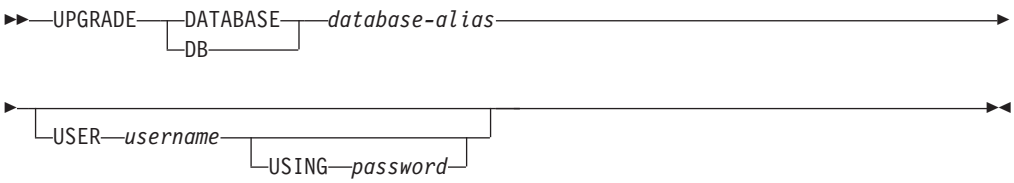
Authorization

sysadm

Required connection

This command establishes a database connection.

Command syntax



Command parameters

DATABASE *database-alias*
Specifies the alias of the database to be upgraded to the currently installed version of the database manager.

USER *username*
Identifies the user name under which the database is to be upgraded.

USING *password*
The password used to authenticate the user name. If the password is omitted, but a user name was specified, the user is prompted to enter it.

Examples

The following example upgrades the database cataloged under the database alias `sales`:

```
db2 UPGRADE DATABASE sales
```

Usage notes

This command will only upgrade a database to a newer version, and cannot be used to convert an upgraded database to its previous version.

The database must be cataloged before upgrade.

If an error occurs during upgrade, it might be necessary to issue the TERMINATE command before attempting the suggested user response. For example, if a log full error occurs during upgrade (SQL1704: Database upgrade failed. Reason code ″3″.), it will be necessary to issue the TERMINATE command before increasing the values of the database configuration parameters LOGPRIMARY and LOGFILSIZ. The CLP must refresh its database directory cache if the upgrade failure occurs after the database has already been relocated (which is likely to be the case when a ″log full″ error returns).

QUIESCE

Scope

QUIESCE DATABASE results in all objects in the database being in the quiesced mode. Only the allowed user/group and *sysadm*, *sysmaint*, *dbadm*, or *sysctrl* will be able to access the database or its objects.

Authorization

One of the following:

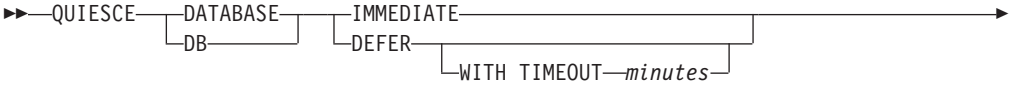
For database level quiesce:

- *sysadm*
- *dbadm*

Required connection

Database

Command syntax



Command parameters

DEFER
Wait for applications until they commit the current unit of work.

WITH TIMEOUT *minutes*
Specifies a time, in minutes, to wait for applications to commit the current unit of work. If no value is specified, in a single-partition database environment, the default value is 10 minutes. In a partitioned database environment the value specified by the **start_stop_time** database manager configuration parameter will be used.

IMMEDIATE
Do not wait for the transactions to be committed, immediately rollback the transactions.

FORCE CONNECTIONS
Force the connections off.

DATABASE
Quiesce the database. All objects in the database will be placed in quiesced mode. Only specified users in specified groups and users with *sysadm*, *sysmaint*, and *sysctrl* authority will be able to access to the database or its objects.

Usage notes

- After QUIESCE DATABASE, users with *sysadm*, *sysmaint*, *sysctrl*, or *dbadm* authority, and GRANT/REVOKE privileges can designate who will be able to connect. This information will be stored permanently in the database catalog tables.

  For example,

  ```
  grant quiesce_connect on database to <username/groupname>
  revoke quiesce_connect on database from <username/groupname>
  ```

QUIESCE TABLESPACES FOR TABLE

Quiesces table spaces for a table. There are three valid quiesce modes: share, intent to update, and exclusive. There are three possible states resulting from the quiesce function:

- Quiesced: SHARE
- Quiesced: UPDATE
- Quiesced: EXCLUSIVE

Scope

In a single-partition environment, this command quiesces all table spaces involved in a load operation in exclusive mode for the duration of the load operation. In a partitioned database environment, this command acts locally on a database partition. It quiesces only that portion of table spaces belonging to the database partition on which the load operation is performed. For partitioned tables, all of the table spaces listed in SYSDATAPARTITIONS.TBSPACEID and SYSDATAPARTITIONS.LONG_TBSPACEID associated with a table and with a

status of normal, attached or detached, (for example, SYSDATAPARTITIONS.STATUS of '', 'A' or 'D', respectively) are quiesced.

Authorization

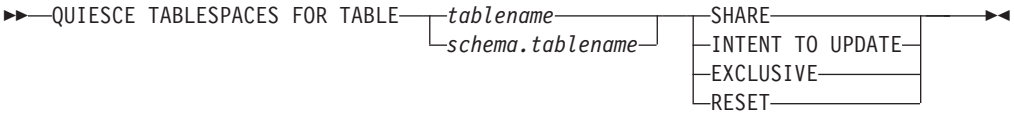
One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

Required connection

Database

Command syntax



Command parameters

TABLE

tablename
Specifies the unqualified table name. The table cannot be a system catalog table.

schema.tablename
Specifies the qualified table name. If *schema* is not provided, the CURRENT SCHEMA will be used. The table cannot be a system catalog table.

SHARE
Specifies that the quiesce is to be in share mode.

When a "quiesce share" request is made, the transaction requests intent share locks for the table spaces and a share lock for the table. When the transaction obtains the locks, the state of the table spaces is changed to QUIESCED SHARE. The state is granted to the quiescer only if there is no conflicting state held by other users. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table, so that the state is persistent. The table cannot be changed while the table spaces for the table are in QUIESCED SHARE state. Other share mode requests to the table and table spaces are allowed. When the transaction commits or rolls back, the locks are released, but the table spaces for the table remain in QUIESCED SHARE state until the state is explicitly reset.

INTENT TO UPDATE
Specifies that the quiesce is to be in intent to update mode.

When a ″quiesce intent to update″ request is made, the table spaces are locked in intent exclusive (IX) mode, and the table is locked in update (U) mode. The state of the table spaces is recorded in the table space table.

EXCLUSIVE
Specifies that the quiesce is to be in exclusive mode.

When a ″quiesce exclusive″ request is made, the transaction requests super exclusive locks on the table spaces, and a super exclusive lock on the table. When the transaction obtains the locks, the state of the table spaces changes to QUIESCED EXCLUSIVE. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table. Since the table spaces are held in super exclusive mode, no other access to the table spaces is allowed. The user who invokes the quiesce function (the quiescer) has exclusive access to the table and the table spaces.

RESET
Specifies that the state of the table spaces is to be reset to normal. A quiesce state cannot be reset if the connection that issued the quiesce request is still active.

Example

Usage notes

This command is not supported for declared temporary tables.

A quiesce is a persistent lock. Its benefit is that it persists across transaction failures, connection failures, and even across system failures (such as power failure, or reboot).

A quiesce is owned by a connection. If the connection is lost, the quiesce remains, but it has no owner, and is called a *phantom quiesce*. For example, if a power outage caused a load operation to be interrupted during the delete phase, the table spaces for the loaded table would be left in delete pending, quiesce exclusive state. Upon database restart, this quiesce would be an unowned (or phantom) quiesce. The removal of a phantom quiesce requires a connection with the same user ID used when the quiesce mode was set.

To remove a phantom quiesce:

1. Connect to the database with the same user ID used when the quiesce mode was set.
2. Use the LIST TABLESPACES command to determine which table space is quiesced.
3. Re-quiesce the table space using the current quiesce state. For example:

Once completed, the new connection owns the quiesce, and the load operation can be restarted.

There is a limit of five quiescers on a table space at any given time.

A quiescer can upgrade the state of a table space from a less restrictive state to a more restrictive one (for example, S to U, or U to X). If a user requests a state lower than one that is already held, the original state is returned. States are not downgraded.

RECOVER DATABASE

Restores and rolls forward a database to a particular point in time or to the end of the logs.

Scope

In a partitioned database environment, this command can only be invoked from the catalog partition. A database recover operation to a specified point in time affects all database partitions that are listed in the `db2nodes.cfg` file. A database recover operation to the end of logs affects the database partitions that are specified. If no partitions are specified, it affects all database partitions that are listed in the `db2nodes.cfg` file.

Authorization

To recover an existing database, one of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

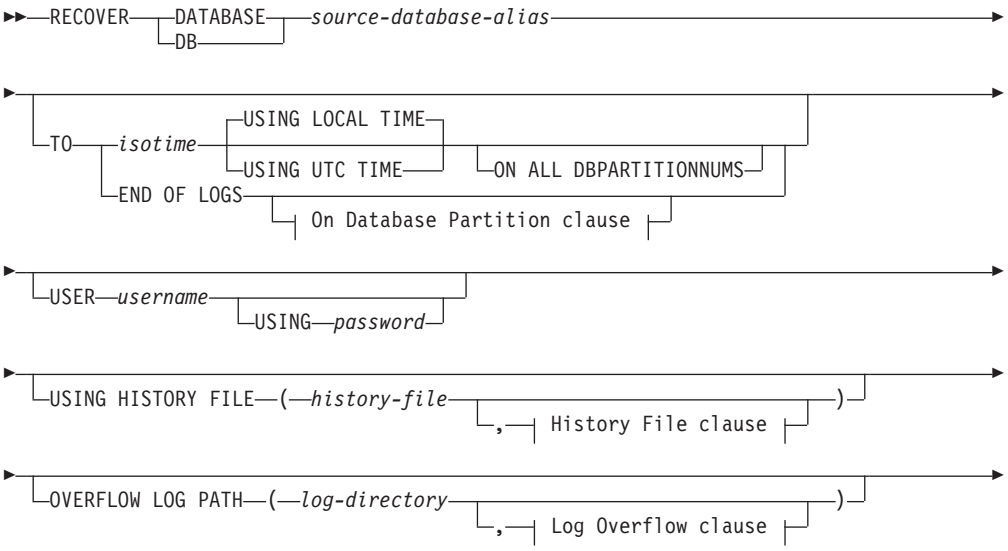
To recover to a new database, one of the following:

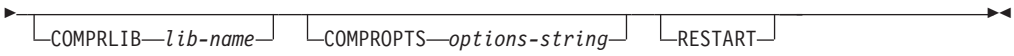
- *sysadm*
- *sysctrl*

Required connection

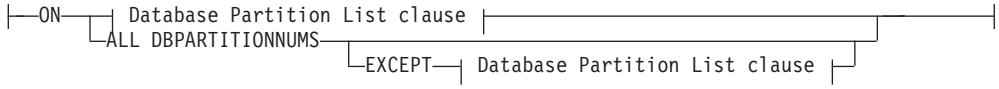
To recover an existing database, a database connection is required. This command automatically establishes a connection to the specified database and will release the connection when the recover operation finishes. To recover to a new database, an instance attachment and a database connection are required. The instance attachment is required to create the database.

Command syntax

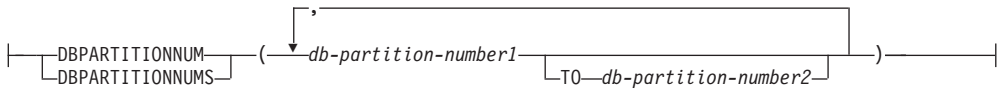




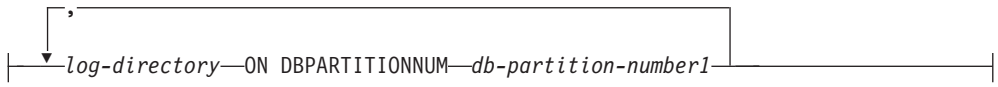
On Database Partition clause:



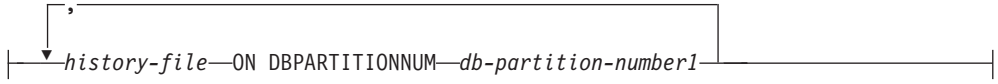
Database Partition List clause:



Log Overflow clause:



History File clause:



Command parameters

DATABASE *database-alias*
The alias of the database that is to be recovered.

USER *username*
The user name under which the database is to be recovered.

USING *password*
The password used to authenticate the user name. If the password is omitted, the user is prompted to enter it.

TO

isotime The point in time to which all committed transactions are to be recovered (including the transaction committed precisely at that time, as well as all transactions committed previously).

This value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds). The time stamp in a backup image is based on the local time at which the backup operation started. The CURRENT TIMEZONE special register specifies the difference between UTC and local time at the application server. The difference is represented by a time duration (a decimal number in which the first two digits represent the number of hours, the next two digits represent the number of minutes, and the last two digits

represent the number of seconds). Subtracting CURRENT TIMEZONE from a local time converts that local time to UTC.

USING LOCAL TIME
Specifies the point in time to which to recover. This option allows the user to recover to a point in time that is the server's local time rather than UTC time. This is the default option.

Note:

1. If the user specifies a local time for recovery, all messages returned to the user will also be in local time. All times are converted on the server, and in partitioned database environments, on the catalog database partition.
2. The timestamp string is converted to UTC on the server, so the time is local to the server's time zone, not the client's. If the client is in one time zone and the server in another, the server's local time should be used. This is different from the local time option from the Control Center, which is local to the client.
3. If the timestamp string is close to the time change of the clock due to daylight saving time, it is important to know if the stop time is before or after the clock change, and specify it correctly.

USING UTC TIME
Specifies the point in time to which to recover.

END OF LOGS
Specifies that all committed transactions from all online archive log files listed in the database configuration parameter **logpath** are to be applied.

ON ALL DBPARTITIONNUMS
Specifies that transactions are to be rolled forward on all database partitions specified in the `db2nodes.cfg` file. This is the default if a database partition clause is not specified.

EXCEPT
Specifies that transactions are to be rolled forward on all database partitions specified in the `db2nodes.cfg` file, except those specified in the database partition list.

ON DBPARTITIONNUM | ON DBPARTITIONNUMS
Roll the database forward on a set of database partitions.

db-partition-number1
Specifies a database partition number in the database partition list.

TO *db-partition-number2*
Specifies the second database partition number, so that all database partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

USING HISTORY FILE *history-file*

history-file **ON DBPARTITIONNUM**
In a partitioned database environment, allows a different history file

OVERFLOW LOG PATH *log-directory*
Specifies an alternate log path to be searched for archived logs during recovery. Use this parameter if log files were moved to a location other than that specified by the **logpath** database configuration parameter. In a partitioned database environment, this is the (fully qualified) default

overflow log path *for all database partitions*. A relative overflow log path can be specified for single-partition databases.

The OVERFLOW LOG PATH command parameter will overwrite the value (if any) of the database configuration parameter **overflowlogpath**.

COMPRLIB *lib-name*
Indicates the name of the library to be used to perform the decompression. The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library cannot be loaded, the restore operation will fail.

COMPROPTS *options-string*
Describes a block of binary data that is passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so any issues of byte reversal or code page conversion are handled by the decompression library. If the first character of the data block is "@", the remainder of the data is interpreted by the DB2 database system as the name of a file residing on the server. The DB2 database system will then replace the contents of *string* with the contents of this file and pass the new value to the initialization routine instead. The maximum length for the string is 1 024 bytes.

RESTART
The RESTART keyword can be used if a prior RECOVER operation was interrupted or otherwise did not complete. Starting in V9.1, a subsequent RECOVER command will attempt to continue the previous RECOVER, if possible. Using the RESTART keyword forces RECOVER to start with a fresh restore and then rollforward to the PIT specified.

log-directory **ON DBPARTITIONNUM**
In a partitioned database environment, allows a different log path to override the default overflow log path for a specific database partition.

Examples

In a single-partition database environment, where the database being recovered currently exists, and the most recent version of the history file is available in the **dftdbpath**:

1. To use the latest backup image and rollforward to the end of logs using all default values:

   ```
   RECOVER DB SAMPLE
   ```

2. To recover the database to a PIT, issue the following. The most recent image that can be used will be restored, and logs applied until the PIT is reached.

   ```
   RECOVER DB SAMPLE TO  2001-12-31-04.00.00
   ```

3. To recover the database using a saved version of the history file, issue the following. For example, if the user needs to recover to an extremely old PIT which is no longer contained in the current history file, the user will have to provide a version of the history file from this time period. If the user has saved a history file from this time period, this version can be used to drive the recover.

   ```
   RECOVER DB SAMPLE TO  1999-12-31-04.00.00
      USING HISTORY FILE (/home/user/old1999files/db2rhist.asc)
   ```

In a single-partition database environment, where the database being recovered does not exist, you must use the USING HISTORY FILE clause to point to a history file.

1. If you have not made any backups of the history file, so that the only version available is the copy in the backup image, the recommendation is to issue a RESTORE followed by a ROLLFORWARD. However, to use RECOVER, you would first have to extract the history file from the image to some location, for example `/home/user/oldfiles/db2rhist.asc`, and then issue this command. (This version of the history file does not contain any information about log files that are required for rollforward, so this history file is not useful for RECOVER.)

   ```
   RECOVER DB SAMPLE TO END OF LOGS
      USING HISTORY FILE (/home/user/fromimage/db2rhist.asc)
   ```

2. If you have been making periodic or frequent backup copies of the history, the USING HISTORY FILE clause should be used to point to this version of the history file. If the file is `/home/user/myfiles/db2rhist.asc`, issue the command:

   ```
   RECOVER DB SAMPLE TO PIT
      USING HISTORY FILE (/home/user/myfiles/db2rhist.asc)
   ```

   (In this case, you can use any copy of the history file, not necessarily the latest, as long as it contains a backup taken before the point-in-time (PIT) requested.)

In a partitioned database environment, where the database exists on all database partitions, and the latest history file is available on **dftdbpath** on all database partitions:

1. To recover the database to a PIT on all nodes. DB2 will verify that the PIT is reachable on all nodes before starting any restore operations.

   ```
   RECOVER DB SAMPLE TO  2001-12-31-04.00.00
   ```

2. To recover the database to this PIT on all nodes. DB2 will verify that the PIT is reachable on all nodes before starting any restore operations. The RECOVER operation on each node is identical to a single-partition RECOVER.

   ```
   RECOVER DB SAMPLE TO END OF LOGS
   ```

3. Even though the most recent version of the history file is in the **dftdbpath**, you might want to use several specific history files. Unless otherwise specified, each database partition will use the history file found locally at `/home/user/oldfiles/db2rhist.asc`. The exceptions are nodes 2 and 4. Node 2 will use: `/home/user/node2files/db2rhist.asc`, and node 4 will use: `/home/user/node4files/db2rhist.asc`.

   ```
   RECOVER DB SAMPLE TO  1999-12-31-04.00.00
      USING HISTORY FILE (/home/user/oldfiles/db2rhist.asc,
         /home/user/node2files/db2rhist.asc ON DBPARTITIONNUM 2,
         /home/user/node4files/db2rhist.asc ON DBPARTITIONNUM 4)
   ```

4. It is possible to recover a subset of nodes instead of all nodes, however a PIT RECOVER can not be done in this case, the recover must be done to EOL.

   ```
   RECOVER DB SAMPLE TO END OF LOGS ON DBPARTITIONNUMS(2 TO 4, 7, 9)
   ```

In a partitioned database environment, where the database does not exist:

1. If you have not made any backups of the history file, so that the only version available is the copy in the backup image, the recommendation is to issue a RESTORE followed by a ROLLFORWARD. However, to use RECOVER, you would first have to extract the history file from the image to some location, for example, `/home/user/oldfiles/db2rhist.asc`, and then issue this command.

(This version of the history file does not contain any information about log files that are required for rollforward, so this history file is not useful for the recover.)

```
RECOVER DB SAMPLE TO PIT
   USING HISTORY FILE (/home/user/fromimage/db2rhist.asc)
```

2. If you have been making periodic or frequent backup copies of the history, the USING HISTORY FILE clause should be used to point to this version of the history file. If the file is /home/user/myfiles/db2rhist.asc, you can issue the following command:

```
RECOVER DB SAMPLE TO END OF LOGS
   USING HISTORY FILE (/home/user/myfiles/db2rhist.asc)
```

Usage notes

- Recovering a database might require a load recovery using tape devices. If prompted for another tape, the user can respond with one of the following:

  c Continue. Continue using the device that generated the warning message (for example, when a new tape has been mounted).

  d Device terminate. Stop using the device that generated the warning message (for example, when there are no more tapes).

  t Terminate. Terminate all devices.
- If there is a failure during the restore portion of the recover operation, you can reissue the RECOVER DATABASE command. If the restore operation was successful, but there was an error during the rollforward operation, you can issue a ROLLFORWARD DATABASE command, since it is not necessary (and it is time-consuming) to redo the entire recover operation.
- In a partitioned database environment, if there is an error during the restore portion of the recover operation, it is possible that it is only an error on a single database partition. Instead of reissuing the RECOVER DATABASE command, which restores the database on all database partitions, it is more efficient to issue a RESTORE DATABASE command for the database partition that failed, followed by a ROLLFORWARD DATABASE command.

REDISTRIBUTE DATABASE PARTITION GROUP

Redistributes data across the database partitions in a database partition group. The target distribution of data can be uniform (default) or user specified to meet specific system requirements.

The REDISTRIBUTE DATABASE PARTITION GROUP command redistributes data across all partitions in a database partition group. This affects all objects present in the database partition group and cannot be restricted to one object alone.

Scope

This command affects all database partitions in the database partition group.

Authorization

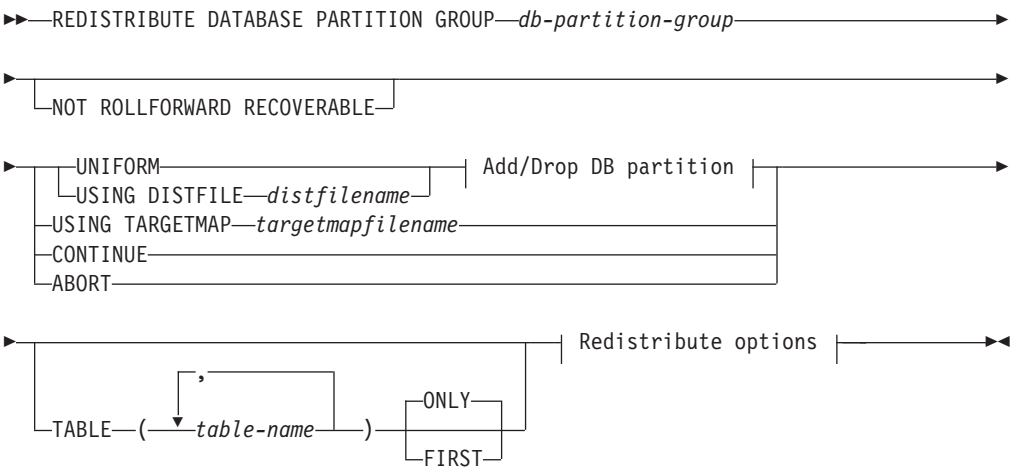
One of the following authorities is required:

- SYSADM
- SYSCTRL
- DBADM

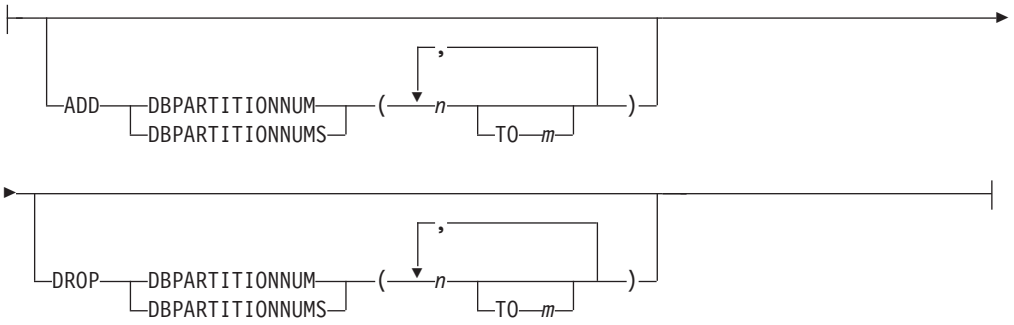
In addition, one of the following groups of authorizations is also required:

- DELETE, INSERT, and SELECT privileges on all tables in the database partition group being redistributed
- DATAACCESS authority

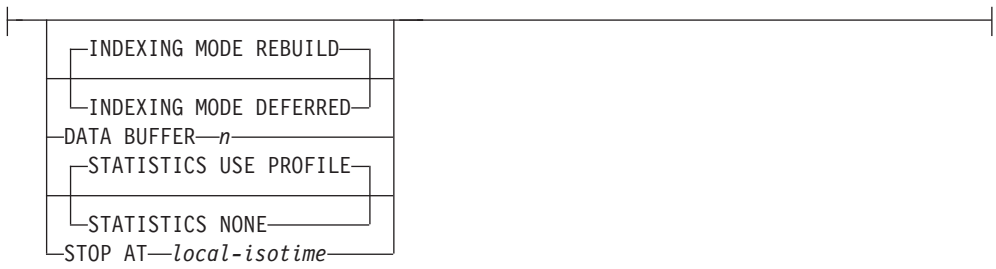
Command syntax



Add/Drop DB partition:



Redistribute options:



Command parameters

DATABASE PARTITION GROUP *db-partition-group*
The name of the database partition group. This one-part name identifies a

database partition group described in the SYSCAT.DBPARTITIONGROUPS catalog table. The database partition group cannot currently be undergoing redistribution.

Note: Tables in the IBMCATGROUP and the IBMTEMPGROUP database partition groups cannot be redistributed.

NOT ROLLFORWARD RECOVERABLE
When this option is used, the REDISTRIBUTE DATABASE PARTITION GROUP command is not roll forward recoverable.

- Data is moved in bulk instead of by internal insert and delete operations. This reduces the number of times that a table must be scanned and accessed, which results in better performance.
- Log records are no longer required for each of the insert and delete operations. This means that you no longer need to manage large amounts of active log space and log archiving space in your system when performing data redistribution. This is particularly beneficial if, in the past, large active log space and storage requirements forced you to break a single data redistribution operation into multiple smaller redistribution tasks, which might have resulted in even more time required to complete the end-to-end data redistribution operation.
- When using the REDISTRIBUTE DATABASE PARTITION GROUP command with the **NOT ROLLFORWARD RECOVERABLE** option, the redistribute operation uses the **INDEXING MODE DEFERRED** option for tables that contain XML columns. If a table does not contain an XML column, the redistribute operation uses the indexing mode specified when issuing the command.

When this option is *not* used, extensive logging of all row movement is performed such that the database can be recovered later in the event of any interruptions, errors, or other business need.

UNIFORM
Specifies that the data is uniformly distributed across hash partitions (that is, every hash partition is assumed to have the same number of rows), but the same number of hash partitions do not map to each database partition. After redistribution, all database partitions in the database partition group have approximately the same number of hash partitions.

USING DISTFILE *distfilename*
If the distribution of distribution key values is skewed, use this option to achieve a uniform redistribution of data across the database partitions of a database partition group.

Use the *distfilename* to indicate the current distribution of data across the 32 768 hash partitions.

Use row counts, byte volumes, or any other measure to indicate the amount of data represented by each hash partition. The utility reads the integer value associated with a partition as the weight of that partition. When a *distfilename* is specified, the utility generates a target distribution map that it uses to redistribute the data across the database partitions in the database partition group as uniformly as possible. After the redistribution, the weight of each database partition in the database partition group is approximately the same (the weight of a database partition is the sum of the weights of all hash partitions that map to that database partition).

For example, the input distribution file might contain entries as follows:

```
10223
1345
112000
0
100
...
```

In the example, hash partition 2 has a weight of 112000, and partition 3 (with a weight of 0) has no data mapping to it at all.

The *distfilename* should contain 32 768 positive integer values in character format. The sum of the values should be less than or equal to 4 294 967 295.

USING TARGETMAP *targetmapfilename*
The file specified in *targetmapfilename* is used as the target distribution map. Data redistribution is done according to this file.

If a database partition, included in the target map, is not in the database partition group, an error is returned. Issue ALTER DATABASE PARTITION GROUP ADD DBPARTITIONNUM statement before running REDISTRIBUTE DATABASE PARTITION GROUP command.

If a database partition, excluded from the target map, *is* in the database partition group, that database partition will not be included in the partitioning. Such a database partition can be dropped using ALTER DATABASE PARTITION GROUP DROP DBPARTITIONNUM statement either before or after the REDISTRIBUTE DATABASE PARTITION GROUP command.

CONTINUE
Continues a previously failed or stopped REDISTRIBUTE DATABASE PARTITION GROUP operation. If none occurred, an error is returned.

ABORT
Aborts a previously failed or stopped REDISTRIBUTE DATABASE PARTITION GROUP operation. If none occurred, an error is returned.

ADD

DBPARTITIONNUM *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be added into the database partition group. Any specified partition must not already be defined in the database partition group (SQLSTATE 42728). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with ADD DBPARTITIONNUM clause specified.

DBPARTITIONNUMS *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be added into the database partition group. Any specified partition must not already be defined in the database partition group (SQLSTATE 42728). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with ADD DBPARTITIONNUM clause specified.

Note: When a database partition is added using this option, containers for table spaces are based on the containers of the corresponding table space on the lowest numbered existing partition in the database partition group. If this would result in a naming conflict among containers, which could happen if the new partitions are on the same physical machine as existing containers, this option should not be used. Instead, the ALTER DATABASE PARTITION GROUP statement should be used with the WITHOUT TABLESPACES option prior to issuing the REDISTRIBUTE DATABASE PARTITION GROUP command. Table space containers can then be created manually specifying appropriate names.

DROP

DBPARTITIONNUM *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be dropped from the database partition group. Any specified partition must already be defined in the database partition group (SQLSTATE 42729). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with the DROP DBPARTITIONNUM clause specified.

DBPARTITIONNUMS *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be dropped from the database partition group. Any specified partition must already be defined in the database partition group (SQLSTATE 42729). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with the DROP DBPARTITIONNUM clause specified.

TABLE *tablename*
Specifies a table order for redistribution processing.

ONLY If the table order is followed by the **ONLY** keyword (which is the default), then, only the specified tables will be redistributed. The remaining tables can be later processed by subsequent REDISTRIBUTE CONTINUE commands. This is the default.

FIRST If the table order is followed by the **FIRST** keyword, then, the specified tables will be redistributed with the given order and the remaining tables in the database partition group will be redistributed with random order.

INDEXING MODE
This parameter specifies how indexes are maintained during redistribution when the **NOT ROLLFORWARD RECOVERABLE** option is specified. Valid values are:

REBUILD
Indexes will be rebuilt from scratch. Indexes do not have to be valid to use this option. As a result of using this option, index pages will be clustered together on disk.

DEFERRED
Redistribute will not attempt to maintain any indexes. Indexes will

be marked as needing a refresh. The first access to such indexes may force a rebuild, or indexes may be rebuilt when the database is restarted.

Note: For non-MDC tables, if there are invalid indexes on the tables, the REDISTRIBUTE DATABASE PARTITION GROUP command automatically rebuilds them if you do not specify **INDEXING MODE DEFERRED**. For an MDC table, even if you specify **INDEXING MODE DEFERRED**, a composite index that is invalid is rebuilt before table redistribution begins because the utility needs the composite index to process an MDC table.

DATA BUFFER *n*
Specifies the number of 4 KB pages to use as buffered space for transferring data within the utility. If the value specified is lower than the minimum supported value, the minimum value is used and no warning is returned. If a DATA BUFFER value is not specified, an intelligent default is calculated by the utility at runtime at the beginning of processing each table. Specifically, the default is to use 50% of the memory available in the utility heap at the time redistribution of the table begins and to take into account various table properties as well.

This memory is allocated directly from the utility heap, whose size can be modified through the **util_heap_sz** database configuration parameter. Beginning in version 9.5, the value of the DATA BUFFER option of the REDISTRIBUTE DATABASE PARTITION GROUP command can temporarily exceed **util_heap_sz** if more memory is available in the system.

STOP AT *local-isotime*
When this option is specified, before beginning data redistribution for each table, the *local-isotime* is compared with the current local timestamp. If the specified *local-isotime* is equal to or earlier than the current local timestamp, the utility stops with a warning message. Data redistribution processing of tables in progress at the stop time will complete without interruption. No new data redistribution processing of tables begins. The unprocessed tables can be redistributed using the **CONTINUE** option. This *local-isotime* value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds) expressed in local time.

STATISTICS
This option specifies that the utility should collect statistics for the tables that have a statistics profile. Specifying this option is more efficient than separately issuing the RUNSTATS command after the data redistribution is completed.

USE PROFILE
Statistics will be collected for the tables with a statistics profile. For tables without a statistics profile, nothing will be done. This is the default.

NONE
Statistics will not be collected for tables.

Examples: Redistribute steps

You may want to add or drop node from node group. Following is the step for adding new node to a node group and redistribute the data. Added database

partition is not in the distribution map, but the containers for the table spaces in the database partition group have been created; the database partition is added to the distribution map when a redistribute database partition group operation has completed successfully.

1. Identify the nodegroups that will require redistribution. In this document, the node group that needs to be redistributed is “sampleNodegrp”.
2. Identify objects that should be disabled or removed before redistribute .
   a. Replicate MQTs: This type of MQT is not supported as part of the REDISTRIBUTE utility. They need to be dropped before running redistribute and recreated afterward.

      ```
      SELECT tabschema, tabname
        FROM syscat.tables
        WHERE partition_mode = 'R'
      ```

   b. Write-to-table event monitors: You should disable any automatically activated write-to-table event monitors that have a table that resides in the database partition group to be redistributed.

      ```
      SELECT distinct evmonname
        FROM syscat.eventtables E
        JOIN syscat.tables T on T.tabname = E.tabname
          AND T.tabschema = E.tabschema
        JOIN syscat.tablespaces S on S.tbspace = T.tbspace
          AND S.ngname = 'sampleNodegrp'
      ```

   c. Explain tables: It is recommended to create the explain tables in a single partition nodegroup. However, if they are defined in a nodegroup that requires redistribution, you may consider dropping them before the redistribute and redefining them once redistribute is complete, if the data generated to date does not need to be maintained.
   d. Table access mode and load state: Ensure that all tables in the node groups to be redistributed are in full access mode and have no load pending or load in progress state.

      ```
      SELECT DISTINCT TRIM(T.OWNER) || \'.\' || TRIM(T.TABNAME) AS NAME, T.ACCESS_MODE, A.LOAD_S
       FROM SYSCAT.TABLES T, SYSCAT.DBPARTITIONGROUPS N, SYSIBMADM.ADMINTABINFO A
       WHERE T.PMAP_ID = N.PMAP_ID
        AND A.TABSCHEMA = T.OWNER
        AND A.TABNAME = T.TABNAME
        AND N.DBPGNAME = 'sampleNodegrp'
        AND (T.ACCESS_MODE <> 'F' OR A.LOAD_STATUS IS NOT NULL)
      ```

   e. Statistics profiles: Table statistics can be updated as part of the redistribution process if a statistics profile is defined for the table. Having the REDISTRIBUTE utility update a table’s statistics reduces I/O as all the data is scanned for the redistribute and no additional scan of the data is needed for RUNSTATS.

      ```
      RUNSTATS on table schema.table
        USE PROFILE runstats_profile
        SET PROFILE ONLY
      ```

3. Review the database configuration. **util_heap_sz** is critical to the data movement processing between database partitions – allocate as much memory as possible to **util_heap_sz** for the duration of the redistribution. Sufficient **sortheap** is required, if index rebuild is done as part of the redistribution. Increase **util_heap_sz** and **sortheap** as necessary to improve redistribute performance.
4. Retrieve the database configuration settings to be used for the new database partitions. When adding database partitions, a default database configuration is used. As a result, it’s important to update the database configuration on the new nodes before the REDISTRIBUTE command is issued to ensure that the configuration is balanced across the entire warehouse.

```
SELECT name,
  CASE WHEN deferred_value_flags = 'AUTOMATIC'
    THEN deferred_value_flags
    ELSE substr(deferred_value,1,20)
    END
  AS deferred_value
  FROM sysibmadm.dbcfg
  WHERE dbpartitionnum = existing-node
    AND deferred_value != ''
    AND name NOT IN ('hadr_local_host','hadr_local_svc','hadr_peer_window',
                 'hadr_remote_host','hadr_remote_inst','hadr_remote_svc',
                 'hadr_syncmode','hadr_timeout','backup_pending','codepage',
                 'codeset','collate_info','country','database_consistent',
                 'database_level','hadr_db_role','log_retain_status',
                 'loghead','logpath','multipage_alloc','numsegs','pagesize',
                 'release','restore_pending','restrict_access',
                 'rollfwd_pending','territory','user_exit_status',
                 'number_compat','varchar2_compat','database_memory')
```

5. Backup the database (or the table spaces in nodegroups that will be redistributed), before starting the redistribution process to ensure a recent recovery point.
6. Define the new data BCUs in DB2 by updating the `db2nodes.cfg` file and adding the new data BCU database partition specifications and define the new database partitions to DB2 using the ADD NODE WITHOUT TABLESPACES command.

   ```
   db2start nodenum x export DB2NODE=x
   db2 add node without tablespaces
   db2stop nodenum x
   ```

   Note: If it is not the first logical port on the data BCU, then execute a start and stop of the first logical port number before and after the above sequence of commands for subsequent logical ports.
7. Define system temporary table space containers on the newly defined database partitions.

   ```
   ALTER TABLESPACE tablespace_name
     ADD container_information
     ON dbpartitionnums (x to y)
   ```
8. Add the new logical database partitions to the database partition groups that span the data BCUs.

   ```
   ALTER DATABASE PARTITION GROUP partition_group_name
     ADD dbpartitionnums (x to y)
     WITHOUT TABLESPACES
   ```
9. Define permanent data table space containers on the newly defined database partitions.

   ```
   ALTER TABLESPACE tablespace_name
     ADD container_information
     ON dbpartitionnums (x to y)
   ```
10. Apply the database configuration settings retrieved in step 4 to the new database partitions (or issue a single UPDATE DB CFG command against all database partitions using the new DB2 9.5 single view of configuration support).
11. Capture the definition of and then drop any replicated MQTs existing in the database partition groups to be redistributed.

    ```
    db2look -d dbname -e -z
      schema -t replicated_MQT_table_names
      -o repMQTs.clp
    ```
12. Disable any write-to-table event monitors that exist in the database partition groups to be redistributed.

```
SET EVENT MONITOR monitor_name STATE 0
```

13. Run the REDISTRIBUTE utility to redistribute uniformly across all database partitions. Following shows the simple redistribute command:

```
REDISTRIBUTE DATABASE PARTITION GROUP sampleNodegrp
  NOT ROLLFORWARD RECOVERABLE uniform;
```

User also should consider specifying a table list as input to the REDISTRIBUTE command to enforce the order that the tables will be processed. The REDISTRIBUTE utility will move the data (compressed and compacted). Optionally, indexes will be rebuilt and statistics updated if statistics profiles are defined. Therefore instead of previous command, the following script can be run:

```
REDISTRIBUTE DATABASE PARTITION GROUP sampleNodegrp
  NOT ROLLFORWARD RECOVERABLE uniform
  TABLE (tab1, tab2,...) FIRST;
```

Consequences of using the NOT ROLLFORWARD RECOVERABLE option

When the REDISTRIBUTE DATABASE PARTITION GROUP command is issued and the **NOT ROLLFORWARD RECOVERABLE** option is specified, a minimal logging strategy is used that minimizes the writing of log records for each moved row. This type of logging is important for the usability of the redistribute operation since an approach that fully logs all data movement could, for large systems, require an impractical amount of active and permanent log space and would generally have poorer performance characteristics. It is important, however, for users to be aware that as a result of this minimal logging model, the REDISTRIBUTE DATABASE PARTITION GROUP command is *not* rollforward recoverable. This means that any operation that results in the database rolling forward through a redistribute operation results in all tables touched by the redistribution operation being left in the UNAVAILABLE state. Such tables can only be dropped, which means there is no way to recover the data in these tables. This is why, for recoverable databases, the REDISTRIBUTE DATABASE PARTITION GROUP utility when issued with the NOT ROLLFORWARD RECOVERABLE option puts all table spaces it touches into the BACKUP PENDING state, forcing the user to backup all redistributed table spaces at the end of a successful redistribute operation. With a backup taken after the redistribution operation, the user should not have a need to rollforward through the redistribute operation itself.

There is one very important consequence of the redistribute utility's lack of rollforward recoverability of which the user should be aware: If the user chooses to allow updates to be made against tables in the database (even tables outside the database partition group being redistributed) while the redistribute operation is running, including the period at the end of redistribute where the table spaces touched by redistribute are being backed up by the user, such updates can be lost in the event of a serious failure, for example, a database container is destroyed. The reason that such updates can be lost is that the redistribute operation is not rollforward recoverable. If it is necessary to restore the database from a backup taken prior to the redistribution operation, then it will not be possible to rollforward through the logs in order to replay the updates that were made during the redistribution operation without also rolling forward through the redistribute which, as was described above, leaves the redistributed tables in the UNAVAILABLE state. Thus, the only thing that can be done in this situation is to restore the database from the backup taken prior to redistribute without rolling forward. Then the redistribute operation can be performed again. Unfortunately, all the updates that occurred during the original redistribute operation are lost.

The importance of this point cannot be overemphasized. In order to be certain that there will be no lost updates during a redistribution operation, one of the following must be true:

- The user avoids making updates during the operation of the REDISTRIBUTE DATABASE PARTITION GROUP command, including the period after the command finishes where the affected table spaces are being backed up.
- Updates that are applied during the redistribute operation come from a repeatable source, meaning that they can be applied again at any time. For example, if the source of updates is data that is stored in a file and the updates are applied during batch processing, then clearly even in the event of a failure requiring a database restore, the updates would not be lost since they could simply be applied again at any time.

With respect to allowing updates to the database during the redistribution operation, the user must decide whether such updates are appropriate or not for their scenario based on whether or not the updates can be repeated after a database restore, if necessary.

Note: Not every failure during operation of the REDISTRIBUTE DATABASE PARTITION GROUP command results in this problem. In fact, most do not. The REDISTRIBUTE DATABASE PARTITION GROUP command is fully restartable, meaning that if the utility fails in the middle of its work, it can be easily continued or aborted with the **CONTINUE** or **ABORT** options. The failures mentioned above are failures that require the user to restore from the backup taken prior to the redistribute operation.

Usage notes

- When the **NOT ROLLFORWARD RECOVERABLE** option is specified and the database is a recoverable database, the first time the utility accesses a table space, it is put into the BACKUP PENDING state. All the tables in that table space will become read-only until the table space is backed-up, which can only be done when all tables in the table space have finished being redistributed.
- When a redistribution operation is running, it produces an event log file containing general information about the redistribution operation and information such as the starting and ending time of each table processed. This event log file is written to:
  - The `homeinst/sqllib/redist` directory on Linux and UNIX operating systems, using the following format for subdirectories and file name: *database-name*`.`*database-partition-group-name*`.`*timestamp*`.log`.
  - The **DB2INSTPROF**`\`*instance*`\redist` directory on Windows operating systems (where **DB2INSTPROF** is the value of the **DB2INSTPROF** registry variable), using the following format for subdirectories and file name: *database-name*`.`*database-partition-group-name*`.`*timestamp*`.log`.
  - The time stamp value is the time when the command was issued.
- This utility performs intermittent COMMITs during processing.
- All packages having a dependency on a table that has undergone redistribution are invalidated. It is recommended to explicitly rebind such packages after the redistribute database partition group operation has completed. Explicit rebinding eliminates the initial delay in the execution of the first SQL request for the invalid package. The redistribute message file contains a list of all the tables that have undergone redistribution.
- By default, the redistribute utility will update the statistics for those tables that have a statistics profile. For the tables without a statistics profile, it is

recommended that you separately update the table and index statistics for these tables by calling the db2Runstats API or by issuing the RUNSTATS command after the redistribute operation has completed.
- Database partition groups containing replicated materialized query tables or tables defined with DATA CAPTURE CHANGES cannot be redistributed.
- Redistribution is not allowed if there are user temporary table spaces with existing declared temporary tables or created temporary tables in the database partition group.
- Options such as **INDEXING MODE** are ignored on tables, on which they do not apply, without warning. For example, **INDEXING MODE** will be ignored on tables without indexes.
- Before starting a redistribute operation, ensure there are no tables in the Load Pending state. Table states can be checked by using the LOAD QUERY command.
- The REDISTRIBUTE DATABASE PARTITION GROUP command might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This command might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

Compatibilities

Tables containing XML columns that use the DB2 Version 9.5 or earlier XML record format cannot be redistributed. Use the ADMIN_MOVE_TABLE stored procedure to migrate the table to the new format.

For compatibility with versions earlier than Version 8:
- The keyword **NODEGROUP** can be substituted for **DATABASE PARTITION GROUP**.

REDISTRIBUTE DATABASE PARTITION GROUP

The REDISTRIBUTE DATABASE PARTITION GROUP command redistributes data across all partitions in a database partition group. This affects all objects present in the database partition group and cannot be restricted to one object alone.

Scope

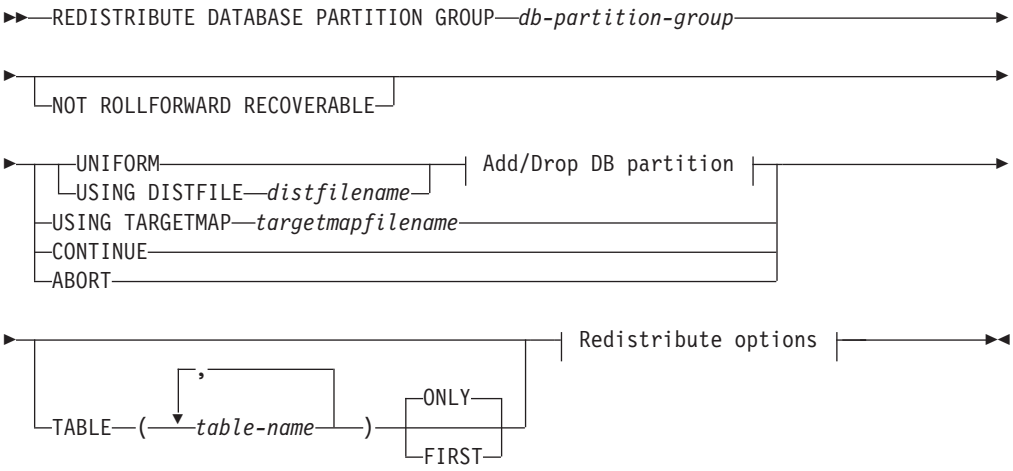
This command affects all database partitions in the database partition group.

Authorization

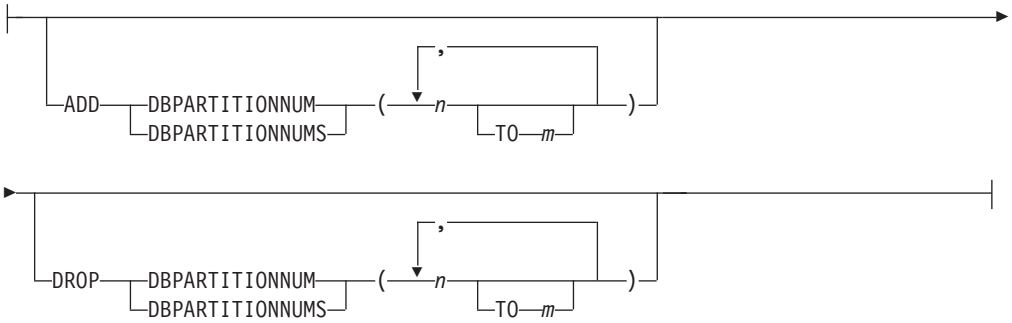
One of the following authorities is required:
- SYSADM
- SYSCTRL
- DBADM

In addition, one of the following groups of authorizations is also required:
- DELETE, INSERT, and SELECT privileges on all tables in the database partition group being redistributed
- DATAACCESS authority

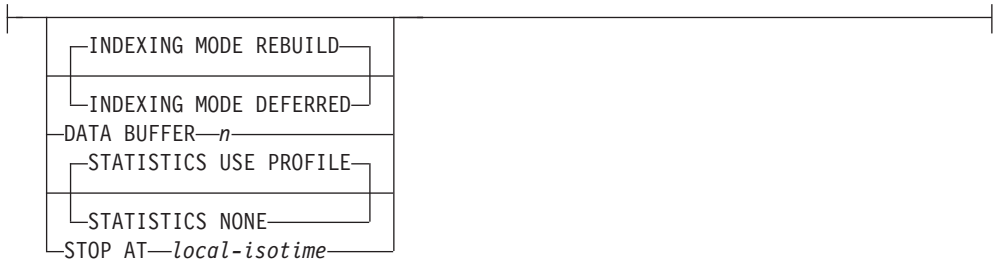
Command syntax



Add/Drop DB partition:



Redistribute options:



Command parameters

DATABASE PARTITION GROUP *db-partition-group*
The name of the database partition group. This one-part name identifies a database partition group described in the SYSCAT.DBPARTITIONGROUPS catalog table. The database partition group cannot currently be undergoing redistribution.

Note: Tables in the IBMCATGROUP and the IBMTEMPGROUP database partition groups cannot be redistributed.

NOT ROLLFORWARD RECOVERABLE

When this option is used, the REDISTRIBUTE DATABASE PARTITION GROUP command is not roll forward recoverable.

- Data is moved in bulk instead of by internal insert and delete operations. This reduces the number of times that a table must be scanned and accessed, which results in better performance.
- Log records are no longer required for each of the insert and delete operations. This means that you no longer need to manage large amounts of active log space and log archiving space in your system when performing data redistribution. This is particularly beneficial if, in the past, large active log space and storage requirements forced you to break a single data redistribution operation into multiple smaller redistribution tasks, which might have resulted in even more time required to complete the end-to-end data redistribution operation.
- When using the REDISTRIBUTE DATABASE PARTITION GROUP command with the **NOT ROLLFORWARD RECOVERABLE** option, the redistribute operation uses the **INDEXING MODE DEFERRED** option for tables that contain XML columns. If a table does not contain an XML column, the redistribute operation uses the indexing mode specified when issuing the command.

When this option is *not* used, extensive logging of all row movement is performed such that the database can be recovered later in the event of any interruptions, errors, or other business need.

UNIFORM

Specifies that the data is uniformly distributed across hash partitions (that is, every hash partition is assumed to have the same number of rows), but the same number of hash partitions do not map to each database partition. After redistribution, all database partitions in the database partition group have approximately the same number of hash partitions.

USING DISTFILE *distfilename*

If the distribution of distribution key values is skewed, use this option to achieve a uniform redistribution of data across the database partitions of a database partition group.

Use the *distfilename* to indicate the current distribution of data across the 32 768 hash partitions.

Use row counts, byte volumes, or any other measure to indicate the amount of data represented by each hash partition. The utility reads the integer value associated with a partition as the weight of that partition. When a *distfilename* is specified, the utility generates a target distribution map that it uses to redistribute the data across the database partitions in the database partition group as uniformly as possible. After the redistribution, the weight of each database partition in the database partition group is approximately the same (the weight of a database partition is the sum of the weights of all hash partitions that map to that database partition).

For example, the input distribution file might contain entries as follows:

```
10223
1345
112000
0
100
...
```

In the example, hash partition 2 has a weight of 112000, and partition 3 (with a weight of 0) has no data mapping to it at all.

The *distfilename* should contain 32 768 positive integer values in character format. The sum of the values should be less than or equal to 4 294 967 295.

USING TARGETMAP *targetmapfilename*
The file specified in *targetmapfilename* is used as the target distribution map. Data redistribution is done according to this file.

If a database partition, included in the target map, is not in the database partition group, an error is returned. Issue ALTER DATABASE PARTITION GROUP ADD DBPARTITIONNUM statement before running REDISTRIBUTE DATABASE PARTITION GROUP command.

If a database partition, excluded from the target map, *is* in the database partition group, that database partition will not be included in the partitioning. Such a database partition can be dropped using ALTER DATABASE PARTITION GROUP DROP DBPARTITIONNUM statement either before or after the REDISTRIBUTE DATABASE PARTITION GROUP command.

CONTINUE
Continues a previously failed or stopped REDISTRIBUTE DATABASE PARTITION GROUP operation. If none occurred, an error is returned.

ABORT
Aborts a previously failed or stopped REDISTRIBUTE DATABASE PARTITION GROUP operation. If none occurred, an error is returned.

ADD

DBPARTITIONNUM *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be added into the database partition group. Any specified partition must not already be defined in the database partition group (SQLSTATE 42728). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with ADD DBPARTITIONNUM clause specified.

DBPARTITIONNUMS *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be added into the database partition group. Any specified partition must not already be defined in the database partition group (SQLSTATE 42728). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with ADD DBPARTITIONNUM clause specified.

Note: When a database partition is added using this option, containers for table spaces are based on the containers of the corresponding table space on the lowest numbered existing partition in the database partition group. If this would result in a naming conflict among containers, which could happen if the new partitions are on the same physical machine as existing containers,

this option should not be used. Instead, the ALTER DATABASE PARTITION GROUP statement should be used with the WITHOUT TABLESPACES option prior to issuing the REDISTRIBUTE DATABASE PARTITION GROUP command. Table space containers can then be created manually specifying appropriate names.

DROP

DBPARTITIONNUM *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be dropped from the database partition group. Any specified partition must already be defined in the database partition group (SQLSTATE 42729). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with the DROP DBPARTITIONNUM clause specified.

DBPARTITIONNUMS *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be dropped from the database partition group. Any specified partition must already be defined in the database partition group (SQLSTATE 42729). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with the DROP DBPARTITIONNUM clause specified.

TABLE *tablename*

Specifies a table order for redistribution processing.

ONLY If the table order is followed by the **ONLY** keyword (which is the default), then, only the specified tables will be redistributed. The remaining tables can be later processed by subsequent REDISTRIBUTE CONTINUE commands. This is the default.

FIRST If the table order is followed by the **FIRST** keyword, then, the specified tables will be redistributed with the given order and the remaining tables in the database partition group will be redistributed with random order.

INDEXING MODE

This parameter specifies how indexes are maintained during redistribution when the **NOT ROLLFORWARD RECOVERABLE** option is specified. Valid values are:

REBUILD

Indexes will be rebuilt from scratch. Indexes do not have to be valid to use this option. As a result of using this option, index pages will be clustered together on disk.

DEFERRED

Redistribute will not attempt to maintain any indexes. Indexes will be marked as needing a refresh. The first access to such indexes may force a rebuild, or indexes may be rebuilt when the database is restarted.

Note: For non-MDC tables, if there are invalid indexes on the tables, the REDISTRIBUTE DATABASE PARTITION GROUP

command automatically rebuilds them if you do not specify **INDEXING MODE DEFERRED**. For an MDC table, even if you specify **INDEXING MODE DEFERRED**, a composite index that is invalid is rebuilt before table redistribution begins because the utility needs the composite index to process an MDC table.

DATA BUFFER *n*
Specifies the number of 4 KB pages to use as buffered space for transferring data within the utility. If the value specified is lower than the minimum supported value, the minimum value is used and no warning is returned. If a DATA BUFFER value is not specified, an intelligent default is calculated by the utility at runtime at the beginning of processing each table. Specifically, the default is to use 50% of the memory available in the utility heap at the time redistribution of the table begins and to take into account various table properties as well.

This memory is allocated directly from the utility heap, whose size can be modified through the **util_heap_sz** database configuration parameter. Beginning in version 9.5, the value of the DATA BUFFER option of the REDISTRIBUTE DATABASE PARTITION GROUP command can temporarily exceed **util_heap_sz** if more memory is available in the system.

STOP AT *local-isotime*
When this option is specified, before beginning data redistribution for each table, the *local-isotime* is compared with the current local timestamp. If the specified *local-isotime* is equal to or earlier than the current local timestamp, the utility stops with a warning message. Data redistribution processing of tables in progress at the stop time will complete without interruption. No new data redistribution processing of tables begins. The unprocessed tables can be redistributed using the **CONTINUE** option. This *local-isotime* value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds) expressed in local time.

STATISTICS
This option specifies that the utility should collect statistics for the tables that have a statistics profile. Specifying this option is more efficient than separately issuing the RUNSTATS command after the data redistribution is completed.

USE PROFILE
Statistics will be collected for the tables with a statistics profile. For tables without a statistics profile, nothing will be done. This is the default.

NONE
Statistics will not be collected for tables.

Examples: Redistribute steps

You may want to add or drop node from node group. Following is the step for adding new node to a node group and redistribute the data. Added database partition is not in the distribution map, but the containers for the table spaces in the database partition group have been created; the database partition is added to the distribution map when a redistribute database partition group operation has completed successfully.

1. Identify the nodegroups that will require redistribution. In this document, the node group that needs to be redistributed is “sampleNodegrp”.

2. Identify objects that should be disabled or removed before redistribute .
   a. Replicate MQTs: This type of MQT is not supported as part of the REDISTRIBUTE utility. They need to be dropped before running redistribute and recreated afterward.

      ```
      SELECT tabschema, tabname
        FROM syscat.tables
        WHERE partition_mode = 'R'
      ```

   b. Write-to-table event monitors: You should disable any automatically activated write-to-table event monitors that have a table that resides in the database partition group to be redistributed.

      ```
      SELECT distinct evmonname
        FROM syscat.eventtables E
        JOIN syscat.tables T on T.tabname = E.tabname
          AND T.tabschema = E.tabschema
        JOIN syscat.tablespaces S on S.tbspace = T.tbspace
          AND S.ngname = 'sampleNodegrp'
      ```

   c. Explain tables: It is recommended to create the explain tables in a single partition nodegroup. However, if they are defined in a nodegroup that requires redistribution, you may consider dropping them before the redistribute and redefining them once redistribute is complete, if the data generated to date does not need to be maintained.
   d. Table access mode and load state: Ensure that all tables in the node groups to be redistributed are in full access mode and have no load pending or load in progress state.

      ```
      SELECT DISTINCT TRIM(T.OWNER) || \'.\' || TRIM(T.TABNAME) AS NAME, T.ACCESS_MODE, A.LOAD_S
       FROM SYSCAT.TABLES T, SYSCAT.DBPARTITIONGROUPS N, SYSIBMADM.ADMINTABINFO A
       WHERE T.PMAP_ID = N.PMAP_ID
        AND A.TABSCHEMA = T.OWNER
        AND A.TABNAME = T.TABNAME
        AND N.DBPGNAME = 'sampleNodegrp'
        AND (T.ACCESS_MODE <> 'F' OR A.LOAD_STATUS IS NOT NULL)
      ```

   e. Statistics profiles: Table statistics can be updated as part of the redistribution process if a statistics profile is defined for the table. Having the REDISTRIBUTE utility update a table's statistics reduces I/O as all the data is scanned for the redistribute and no additional scan of the data is needed for RUNSTATS.

      ```
      RUNSTATS on table schema.table
        USE PROFILE runstats_profile
        SET PROFILE ONLY
      ```

3. Review the database configuration. **util_heap_sz** is critical to the data movement processing between database partitions – allocate as much memory as possible to **util_heap_sz** for the duration of the redistribution. Sufficient **sortheap** is required, if index rebuild is done as part of the redistribution. Increase **util_heap_sz** and **sortheap** as necessary to improve redistribute performance.
4. Retrieve the database configuration settings to be used for the new database partitions. When adding database partitions, a default database configuration is used. As a result, it's important to update the database configuration on the new nodes before the REDISTRIBUTE command is issued to ensure that the configuration is balanced across the entire warehouse.

   ```
   SELECT name,
     CASE WHEN deferred_value_flags = 'AUTOMATIC'
       THEN deferred_value_flags
       ELSE substr(deferred_value,1,20)
       END
     AS deferred_value
     FROM sysibmadm.dbcfg
   ```

```
WHERE dbpartitionnum = existing-node
  AND deferred_value != ''
  AND name NOT IN ('hadr_local_host','hadr_local_svc','hadr_peer_window',
                'hadr_remote_host','hadr_remote_inst','hadr_remote_svc',
                'hadr_syncmode','hadr_timeout','backup_pending','codepage',
                'codeset','collate_info','country','database_consistent',
                'database_level','hadr_db_role','log_retain_status',
                'loghead','logpath','multipage_alloc','numsegs','pagesize',
                'release','restore_pending','restrict_access',
                'rollfwd_pending','territory','user_exit_status',
                'number_compat','varchar2_compat','database_memory')
```

5. Backup the database (or the table spaces in nodegroups that will be redistributed), before starting the redistribution process to ensure a recent recovery point.
6. Define the new data BCUs in DB2 by updating the `db2nodes.cfg` file and adding the new data BCU database partition specifications and define the new database partitions to DB2 using the ADD NODE WITHOUT TABLESPACES command.

   ```
   db2start nodenum x export DB2NODE=x
   db2 add node without tablespaces
   db2stop nodenum x
   ```

   Note: If it is not the first logical port on the data BCU, then execute a start and stop of the first logical port number before and after the above sequence of commands for subsequent logical ports.
7. Define system temporary table space containers on the newly defined database partitions.

   ```
   ALTER TABLESPACE tablespace_name
     ADD container_information
     ON dbpartitionnums (x to y)
   ```
8. Add the new logical database partitions to the database partition groups that span the data BCUs.

   ```
   ALTER DATABASE PARTITION GROUP partition_group_name
     ADD dbpartitionnums (x to y)
     WITHOUT TABLESPACES
   ```
9. Define permanent data table space containers on the newly defined database partitions.

   ```
   ALTER TABLESPACE tablespace_name
     ADD container_information
     ON dbpartitionnums (x to y)
   ```
10. Apply the database configuration settings retrieved in step 4 to the new database partitions (or issue a single UPDATE DB CFG command against all database partitions using the new DB2 9.5 single view of configuration support).
11. Capture the definition of and then drop any replicated MQTs existing in the database partition groups to be redistributed.

    ```
    db2look -d dbname -e -z
      schema -t replicated_MQT_table_names
      -o repMQTs.clp
    ```
12. Disable any write-to-table event monitors that exist in the database partition groups to be redistributed.

    ```
    SET EVENT MONITOR monitor_name STATE 0
    ```
13. Run the REDISTRIBUTE utility to redistribute uniformly across all database partitions. Following shows the simple redistribute command:

    ```
    REDISTRIBUTE DATABASE PARTITION GROUP sampleNodegrp
      NOT ROLLFORWARD RECOVERABLE uniform;
    ```

User also should consider specifying a table list as input to the REDISTRIBUTE command to enforce the order that the tables will be processed. The REDISTRIBUTE utility will move the data (compressed and compacted). Optionally, indexes will be rebuilt and statistics updated if statistics profiles are defined. Therefore instead of previous command, the following script can be run:

```
REDISTRIBUTE DATABASE PARTITION GROUP sampleNodegrp
  NOT ROLLFORWARD RECOVERABLE uniform
  TABLE (tab1, tab2,...) FIRST;
```

Consequences of using the NOT ROLLFORWARD RECOVERABLE option

When the REDISTRIBUTE DATABASE PARTITION GROUP command is issued and the **NOT ROLLFORWARD RECOVERABLE** option is specified, a minimal logging strategy is used that minimizes the writing of log records for each moved row. This type of logging is important for the usability of the redistribute operation since an approach that fully logs all data movement could, for large systems, require an impractical amount of active and permanent log space and would generally have poorer performance characteristics. It is important, however, for users to be aware that as a result of this minimal logging model, the REDISTRIBUTE DATABASE PARTITION GROUP command is *not* rollforward recoverable. This means that any operation that results in the database rolling forward through a redistribute operation results in all tables touched by the redistribution operation being left in the UNAVAILABLE state. Such tables can only be dropped, which means there is no way to recover the data in these tables. This is why, for recoverable databases, the REDISTRIBUTE DATABASE PARTITION GROUP utility when issued with the NOT ROLLFORWARD RECOVERABLE option puts all table spaces it touches into the BACKUP PENDING state, forcing the user to backup all redistributed table spaces at the end of a successful redistribute operation. With a backup taken after the redistribution operation, the user should not have a need to rollforward through the redistribute operation itself.

There is one very important consequence of the redistribute utility's lack of rollforward recoverability of which the user should be aware: If the user chooses to allow updates to be made against tables in the database (even tables outside the database partition group being redistributed) while the redistribute operation is running, including the period at the end of redistribute where the table spaces touched by redistribute are being backed up by the user, such updates can be lost in the event of a serious failure, for example, a database container is destroyed. The reason that such updates can be lost is that the redistribute operation is not rollforward recoverable. If it is necessary to restore the database from a backup taken prior to the redistribution operation, then it will not be possible to rollforward through the logs in order to replay the updates that were made during the redistribution operation without also rolling forward through the redistribute which, as was described above, leaves the redistributed tables in the UNAVAILABLE state. Thus, the only thing that can be done in this situation is to restore the database from the backup taken prior to redistribute without rolling forward. Then the redistribute operation can be performed again. Unfortunately, all the updates that occurred during the original redistribute operation are lost.

The importance of this point cannot be overemphasized. In order to be certain that there will be no lost updates during a redistribution operation, one of the following must be true:

- The user avoids making updates during the operation of the REDISTRIBUTE DATABASE PARTITION GROUP command, including the period after the command finishes where the affected table spaces are being backed up.
- Updates that are applied during the redistribute operation come from a repeatable source, meaning that they can be applied again at any time. For example, if the source of updates is data that is stored in a file and the updates are applied during batch processing, then clearly even in the event of a failure requiring a database restore, the updates would not be lost since they could simply be applied again at any time.

With respect to allowing updates to the database during the redistribution operation, the user must decide whether such updates are appropriate or not for their scenario based on whether or not the updates can be repeated after a database restore, if necessary.

Note: Not every failure during operation of the REDISTRIBUTE DATABASE PARTITION GROUP command results in this problem. In fact, most do not. The REDISTRIBUTE DATABASE PARTITION GROUP command is fully restartable, meaning that if the utility fails in the middle of its work, it can be easily continued or aborted with the **CONTINUE** or **ABORT** options. The failures mentioned above are failures that require the user to restore from the backup taken prior to the redistribute operation.

Usage notes

- When the **NOT ROLLFORWARD RECOVERABLE** option is specified and the database is a recoverable database, the first time the utility accesses a table space, it is put into the BACKUP PENDING state. All the tables in that table space will become read-only until the table space is backed-up, which can only be done when all tables in the table space have finished being redistributed.
- When a redistribution operation is running, it produces an event log file containing general information about the redistribution operation and information such as the starting and ending time of each table processed. This event log file is written to:
  - The `homeinst/sqllib/redist` directory on Linux and UNIX operating systems, using the following format for subdirectories and file name: *database-name*`.`*database-partition-group-name*`.`*timestamp*`.log`.
  - The **DB2INSTPROF**`\`*instance*`\redist` directory on Windows operating systems (where **DB2INSTPROF** is the value of the **DB2INSTPROF** registry variable), using the following format for subdirectories and file name: *database-name*`.`*database-partition-group-name*`.`*timestamp*`.log`.
  - The time stamp value is the time when the command was issued.
- This utility performs intermittent COMMITs during processing.
- All packages having a dependency on a table that has undergone redistribution are invalidated. It is recommended to explicitly rebind such packages after the redistribute database partition group operation has completed. Explicit rebinding eliminates the initial delay in the execution of the first SQL request for the invalid package. The redistribute message file contains a list of all the tables that have undergone redistribution.
- By default, the redistribute utility will update the statistics for those tables that have a statistics profile. For the tables without a statistics profile, it is recommended that you separately update the table and index statistics for these tables by calling the db2Runstats API or by issuing the RUNSTATS command after the redistribute operation has completed.

- Database partition groups containing replicated materialized query tables or tables defined with DATA CAPTURE CHANGES cannot be redistributed.
- Redistribution is not allowed if there are user temporary table spaces with existing declared temporary tables or created temporary tables in the database partition group.
- Options such as **INDEXING MODE** are ignored on tables, on which they do not apply, without warning. For example, **INDEXING MODE** will be ignored on tables without indexes.
- Before starting a redistribute operation, ensure there are no tables in the Load Pending state. Table states can be checked by using the LOAD QUERY command.
- The REDISTRIBUTE DATABASE PARTITION GROUP command might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This command might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

Compatibilities

Tables containing XML columns that use the DB2 Version 9.5 or earlier XML record format cannot be redistributed. Use the ADMIN_MOVE_TABLE stored procedure to migrate the table to the new format.

For compatibility with versions earlier than Version 8:

- The keyword **NODEGROUP** can be substituted for **DATABASE PARTITION GROUP**.

REORG INDEXES/TABLE

Reorganizes an index or a table

You can reorganize all indexes defined on a table by rebuilding the index data into unfragmented, physically contiguous pages. Alternatively, you have the option of reorganizing specific indexes on a range partitioned table.

If you specify the CLEANUP ONLY option of the index clause, cleanup is performed without rebuilding the indexes. This command cannot be used against indexes on declared temporary tables or created temporary tables (SQLSTATE 42995).

The table option reorganizes a table by reconstructing the rows to eliminate fragmented data, and by compacting information.

Scope

This command affects all database partitions in the database partition group.

Authorization

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

- *dbadm*
- *sqladm*
- CONTROL privilege on the table.

Required connection

Database

Command syntax

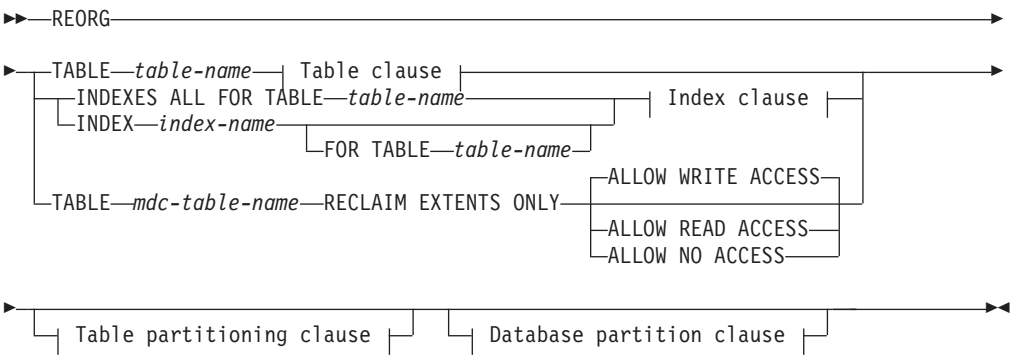
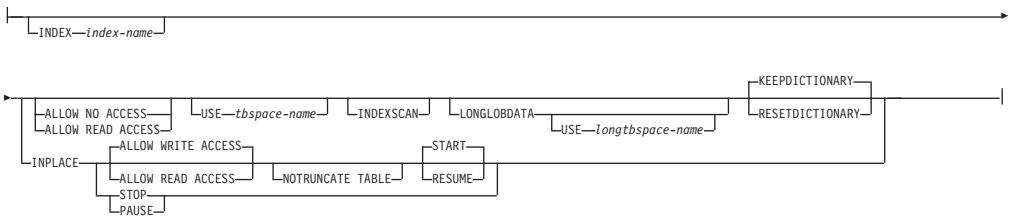


Table clause:



Index clause:

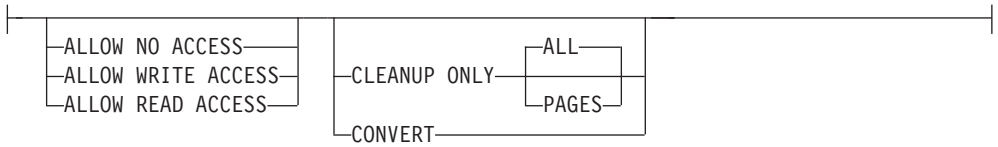
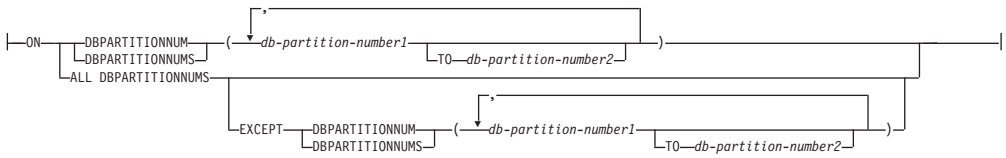


Table partitioning clause:

Database partition clause:



Command parameters

INDEXES ALL FOR TABLE *table-name*
Specifies the table whose indexes are to be reorganized. The table can be in a local or a remote database.

INDEX *index-name*
Specifies an individual index to be reorganized on a partitioned table. Reorganization of individual indexes are *only* supported for nonpartitioned indexes on a partitioned table. This parameter is not supported for block indexes.

FOR TABLE *table-name*
Specifies the table name location of the individual index being reorganized on a partitioned table. This parameter is optional, given that index names are unique across the database.

For REORG INDEXES when the ON DATA PARTITION option is specified, the access clause only applies to the named partition. Users can read from and write to the rest of the table while the indexes on the specified partition are being reorganized. This situation also applies to the default access levels.

ALLOW NO ACCESS
For REORG INDEX and REORG INDEXES, specifies that no other users can access the table or data partition while the indexes are being reorganized.

ALLOW READ ACCESS
For REORG INDEX and REORG INDEXES, specifies that other users can have read-only access to the table or data partition while the indexes are being reorganized. This access level is not supported for REORG INDEXES of a partitioned table unless the CLEANUP ONLY option or the ON DATA PARTITION option is specified.

ALLOW WRITE ACCESS
For REORG INDEX and REORG INDEXES, specifies that other users can read from and write to the table or data partition while the indexes are being reorganized.

This access level is not supported for multidimensional clustering (MDC) tables or extended indexes unless the CLEANUP ONLY option is specified. In addition, this access level is not supported for partitioned tables when the REORG INDEXES command is used unless the CLEANUP ONLY option or the ON DATA PARTITION option is specified.

When ACCESS mode is not specified, it is selected for the table or data partition in the following way:

Table 160. Default table access chosen based on the command, table type and additional parameters specified for the index clause:

Command	Table type	Table partitioning clause	Additional parameters specified for index clause	Default access mode
REORG INDEXES	Nonpartitioned table	Not applicable	Any	ALLOW READ ACCESS

Table 160. Default table access chosen based on the command, table type and additional parameters specified for the index clause: (continued)

Command	Table type	Table partitioning clause	Additional parameters specified for index clause	Default access mode
REORG INDEX	Partitioned table	Not applicable	Any	ALLOW READ ACCESS
REORG INDEXES	Partitioned table	None	None specified	ALLOW NO ACCESS
REORG INDEXES	Partitioned table	ON DATA PARTITION	None specified	ALLOW READ ACCESS
REORG INDEXES	Partitioned table	With or without the ON DATA PARTITION clause	CLEANUP ONLY specified	ALLOW READ ACCESS

CLEANUP ONLY

When CLEANUP ONLY is requested, a cleanup rather than a full reorganization will be done. The indexes will not be rebuilt and any pages freed up will be available for reuse by indexes defined on this table only.

The CLEANUP ONLY PAGES option will search for and free committed pseudo empty pages. A committed pseudo empty page is one where all the keys on the page are marked as deleted and all these deletions are known to be committed. The number of pseudo empty pages in an indexes can be determined by running RUNSTATS and looking at the NUM EMPTY LEAFS column in SYSCAT.INDEXES. The PAGES option will clean the NUM EMPTY LEAFS if they are determined to be committed.

The CLEANUP ONLY ALL option will free committed pseudo empty pages, as well as remove committed pseudo deleted keys from pages that are not pseudo empty. This option will also try to merge adjacent leaf pages if doing so will result in a merged leaf page that has at least PCTFREE free space on the merged leaf page, where PCTFREE is the percent free space defined for the index at index creation time. The default PCTFREE is ten percent. If two pages can be merged, one of the pages will be freed. The number of pseudo deleted keys in an index , excluding those on pseudo empty pages, can be determined by running RUNSTATS and then selecting the NUMRIDS DELETED from SYSCAT.INDEXES. The ALL option will clean the NUMRIDS DELETED and the NUM EMPTY LEAFS if they are determined to be committed.

ALL Specifies that indexes should be cleaned up by removing committed pseudo deleted keys and committed pseudo empty pages.

PAGES

Specifies that committed pseudo empty pages should be removed from the index tree. This will not clean up pseudo deleted keys on pages that are not pseudo empty. Since it is only checking the pseudo empty leaf pages, it is considerably faster than using the ALL option in most cases.

CONVERT
Converts type-1 indexes to type-2 index. If the index is already type 2, this option has no effect.

In Version 9.7, type-1 indexes are discontinued and all indexes that are created are type-2 indexes. As a result, the CONVERT option is deprecated.

All indexes created prior to Version 8 are type-1 indexes. Prior to Version 9.7, all indexes created by Version 8 and later are type-2 indexes, except when you create an index on a table that already has a type-1 index. In this case, the new index was also of type 1. This is no longer the case in Version 9.7 because all indexes created are type 2.

Use the ALLOW READ ACCESS or ALLOW WRITE ACCESS option to allow other transactions either read-only or read-write access to the table while the indexes are being reorganized. While ALLOW READ ACCESS and ALLOW WRITE ACCESS allow access to the table, during the period in which the reorganized copies of the indexes are made available, no access to the table is allowed.

TABLE *mdc-table-name* **RECLAIM EXTENTS ONLY**
Specifies the multidimensional clustering (MDC) table to reorganize to reclaim extents that are not being used. The name or alias in the form: *schema*.*table-name* can be used. The *schema* is the user name under which the table was created. If you omit the schema name, the default schema is assumed.

For REORG TABLE RECLAIM EXTENTS ONLY when the ON DATA PARTITION option is specified, the access clause only applies to the named partition. Users can read from and write to the rest of the table while the extents on the specified partition are being reclaimed. This situation also applies to the default access levels.

ALLOW NO ACCESS
For REORG TABLE RECLAIM EXTENTS ONLY, specifies that no other users can access the table while the extents are being reclaimed.

ALLOW READ ACCESS
For REORG TABLE RECLAIM EXTENTS ONLY, specifies that other users can have read-only access to the table while the extents are being reclaimed.

ALLOW WRITE ACCESS
For REORG TABLE RECLAIM EXTENTS ONLY, specifies that other users can read from and write to the table while the extents are being reclaimed.

TABLE *table-name*
Specifies the table to reorganize. The table can be in a local or a remote database. The name or alias in the form: *schema*.*table-name* can be used. The *schema* is the user name under which the table was created. If you omit the schema name, the default schema is assumed.

For typed tables, the specified table name must be the name of the hierarchy's root table.

You cannot specify an index for the reorganization of a multidimensional clustering (MDC) table. In place reorganization of tables cannot be used for MDC tables.

INDEX *index-name*
Specifies the index to use when reorganizing the table. If you do not specify the fully qualified name in the form: *schema.index-name*, the default schema is assumed. The *schema* is the user name under which the index was created. The database manager uses the index to physically reorder the records in the table it is reorganizing.

For an in place table reorganization, if a clustering index is defined on the table and an index is specified, it must be clustering index. If the in place option is not specified, any index specified will be used. If you do not specify the name of an index, the records are reorganized without regard to order. If the table has a clustering index defined, however, and no index is specified, then the clustering index is used to cluster the table. You cannot specify an index if you are reorganizing an MDC table.

ALLOW NO ACCESS
Specifies that no other users can access the table while the table is being reorganized. When reorganizing a partitioned table with no Table partitioning clause, this value is the default.

ALLOW READ ACCESS
Allow only read access to the table during reorganization. This value is the default for a nonpartitioned table or for a partitioned table using the Table partitioning clause.

When the ON DATA PARTITION option is specified for a REORG TABLE of a range partitioned table, the access clause applies to the named partition only. If all indexes on the table are partitioned, users can read from and write to the rest of the table while the partition is being reorganized and its indexes rebuild. However, if any nonpartitioned indexes are defined for the table, access for the entire table is escalated to ALLOW NO ACCESS. In this situation, the nonpartitioned indexes are rebuilt along with the specified partition.

INPLACE
Reorganizes the table while permitting user access.

In place table reorganization is allowed only on nonpartitioned and non-MDC tables with type-2 indexes, but without extended indexes and with no indexes defined over XML columns in the table. In place table reorganization can only be performed on tables that are at least three pages in size.

In place table reorganization takes place asynchronously, and might not be effective immediately.

ALLOW READ ACCESS
Allow only read access to the table during reorganization.

ALLOW WRITE ACCESS
Allow write access to the table during reorganization. This is the default behavior.

NOTRUNCATE TABLE
Do not truncate the table after in place reorganization. During truncation, the table is S-locked.

START
Start the in place REORG processing. Because this is the default, this keyword is optional.

STOP Stop the in place REORG processing at its current point.

PAUSE
Suspend or pause in place REORG for the time being.

RESUME
Continue or resume a previously paused in place table reorganization. When an online reorganization is resumed and you want the same options as when the reorganization was paused, you must specify those options again while resuming.

USE *tbspace-name*
Specifies the name of a system temporary table space in which to store a temporary copy of the table being reorganized. If you do not provide a table space name, the database manager stores a working copy of the table in the table spaces that contain the table being reorganized.

For an 8KB, 16KB, or 32KB table object, if the page size of the system temporary table space that you specify does not match the page size of the table spaces in which the table data resides, the DB2 database product will try to find a temporary table space of the correct size of the LONG/LOB objects. Such a table space must exist for the reorganization to succeed.

When you have two temporary table spaces of the same page size, and you specify one of them in the USE clause, they will be used in a round robin fashion if there is an index in the table being reorganized. Say you have two table spaces, `tempspace1` and `tempspace2`, both of the same page size and you specify `tempspace1` in the REORG command with the USE option. When you perform REORG the first time, `tempspace1` is used. The second time, `tempspace2` is used. The third time, `tempspace1` is used and so on. To avoid this, you should drop one of the temporary table spaces.

For partitioned tables, the table space is used as temporary storage for the reorganization of data partitions in the table. Reorganization of the entire partitioned table reorganizes a single data partition at a time. The amount of space required is equal to the largest data partition in the table, and not the entire table.

If you do not supply a table space name for a partitioned table, the table space where each data partition is located is used for temporary storage of that data partition. There must be enough free space in each data partition's table space to hold a copy of the data partition.

INDEXSCAN
For a clustering REORG an index scan will be used to re-order table records. Reorganize table rows by accessing the table through an index. The default method is to scan the table and sort the result to reorganize the table, using temporary table spaces as

necessary. Even though the index keys are in sort order, scanning and sorting is typically faster than fetching rows by first reading the row identifier from an index.

LONGLOBDATA
Long field and LOB data are to be reorganized.

This is not required even if the table contains long or LOB columns. The default is to avoid reorganizing these objects because it is time consuming and does not improve clustering. However, running a reorganization with the LONGLOBDATA option on tables with XML columns will reclaim unused space and thereby reduce the size of the XML storage object.

This parameter is required when converting existing LOB data into inlined LOB data.

USE *longtbspace-name*
This is an optional parameter, which can be used to specify the name of a temporary table space to be used for rebuilding long data. If no temporary table space is specified for either the table object or for the long objects, the objects will be constructed in the table space they currently reside. If a temporary table space is specified for the table but this parameter is not specified, then the table space used for base reorg data will be used, unless the page sizes differ. In this situation, the DB2 database system will attempt to choose a temporary container of the appropriate page size to create the long objects in.

If USE *longtbspace-name* is specified, USE *tbspace-name* must also be specified. If it is not, the *longtbspace-name* argument is ignored.

KEEPDICTIONARY
If the `COMPRESS` attribute for the table is `YES` and the table has a compression dictionary then no new dictionary is built. All the rows processed during reorganization are subject to compression using the existing dictionary. If the `COMPRESS` attribute is `YES` and a compression dictionary doesn't exist for the table, a dictionary will only be created (and the table compressed) in this scenario if the table is of a certain size (approximately 1 to 2 MB) and sufficient data exists within this table. If, instead, you explicitly state REORG RESETDICTIONARY, then a dictionary is built as long as there is at least 1 row in the table. If the `COMPRESS` attribute for the table is `NO` and the table has a compression dictionary, then reorg processing will preserve the dictionary and all the rows in the newly reorganized table will be in noncompressed format. It is not possible to compress some data such as LOB data not stored in the base table row.

When the LONGLOBDATA option is not specified, only the table row data is reorganized. The following table describes the behavior of KEEPDICTIONARY syntax in REORG command when the LONGLOBDATA option is not specified.

Table 161. REORG KEEPDICTIONARY

Compress	Dictionary Exists	Result; outcome
Y	Y	Preserve dictionary; rows compressed.
Y	N	Build dictionary; rows compressed

Table 161. REORG KEEPDICTIONARY (continued)

Compress	Dictionary Exists	Result; outcome
N	Y	Preserve dictionary; all rows uncompressed
N	N	No effect; all rows uncompressed

The following table describes the behavior of KEEPDICTIONARY syntax in REORG command when the LONGLOBDATA option is specified.

Table 162. REORG KEEPDICTIONARY when LONGLOBDATA option is specified.

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Compression dictionary	Data compression
Y	Y	Y	Preserve dictionaries.	Existing data is compressed. New data will be compressed.
Y	Y	N	Preserve table row dictionary and create an XML storage object dictionary.	Existing data is compressed. New data will be compressed.
Y	N	Y	Create table row dictionary and preserve the XML dictionary.	Existing data is compressed. New data will be compressed.
Y	N	N	Create table row and XML dictionaries.	Existing data is compressed. New data will be compressed.
N	Y	Y	Preserve table row and XML dictionaries.	Table data is uncompressed. New data will be not be compressed.
N	Y	N	Preserve table row dictionary.	Table data is uncompressed. New data will be not be compressed.
N	N	Y	Preserve XML dictionary.	Table data is uncompressed. New data will be not be compressed.
N	N	N	No effect.	Table data is uncompressed. New data will be not be compressed.

Note:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 V9.7 or later, or if the table is migrated using the ONLINE_TABLE_MOVE stored procedure.

For any reinitialization or truncation of a table (such as for a replace operation), if the compress attribute for the table is NO, the dictionary is discarded if one exists. Conversely, if a dictionary

exists and the compress attribute for the table is YES then a truncation will save the dictionary and not discard it. The dictionary is logged in its entirety for recovery purposes and for future support with data capture changes (that is, replication).

RESETDICTIONARY

If the COMPRESS attribute for the table is YES then a new row compression dictionary is built. All the rows processed during reorganization are subject to compression using this new dictionary. This dictionary replaces any previous dictionary. If the COMPRESS attribute for the table is NO and the table does have an existing compression dictionary then reorg processing will remove the dictionary and all rows in the newly reorganized table will be in noncompressed format. It is not possible to compress some data such as LOB data not stored in the base table row.

If the LONGLOBDATA option is not specified, only the table row data is reorganized. The following table describes the behavior of RESETDICTIONARY syntax in REORG command when the LONGLOBDATA option is not specified.

Table 163. REORG RESETDICTIONARY

Compress	Dictionary Exists	Result; outcome
Y	Y	Build new dictionary*; rows compressed. If DATA CAPTURE CHANGES option is specified on the CREATE TABLE or ALTER TABLE statements, the current dictionary is kept (referred to as the *historical compression dictionary*).
Y	N	Build new dictionary; rows compressed
N	Y	Remove dictionary; all rows uncompressed. If the DATA CAPTURE NONE option is specified on the CREATE TABLE or ALTER TABLE statements, the *historical compression dictionary* is also removed for the specified table.
N	N	No effect; all rows uncompressed

* - If a dictionary exists and the compression attribute is enabled but there currently isn't any data in the table, the RESETDICTIONARY operation will keep the existing dictionary. Rows which are smaller in size than the internal minimum record length and rows which do not demonstrate a savings in record length when an attempt is made to compress them are considered ″insufficient″ in this case.

The following table describes the behavior of RESETDICTIONARY syntax in REORG command when the LONGLOBDATA option is specified.

Table 164. REORG RESETDICTIONARY when LONGLOBDATA option is specified.

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Data dictionary	Data compression
Y	Y	Y	Build dictionaries[2] [3].	Existing data is compressed. New data will be compressed.

Table 164. REORG RESETDICTIONARY when LONGLOBDATA option is specified. (continued)

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Data dictionary	Data compression
Y	Y	N	Build new table row dictionary and create a new XML dictionary[3].	Existing data is compressed. New data will be compressed.
Y	N	Y	Create table row data dictionary and build a new XML dictionary.	Existing data is compressed. New data will be compressed.
Y	N	N	Create dictionaries.	Existing data is compressed. New data will be compressed.
N	Y	Y	Remove dictionaries. Existing and new data is not compressed.	Existing table data is uncompressed. New data will be not be compressed.
N	Y	N	Remove table row dictionary. All data is uncompressed.	Existing table data is uncompressed. New data will be not be compressed.
N	N	Y	Remove XML storage object dictionary.	Existing table data is uncompressed. New data will be not be compressed.
N	N	N	No effect.	Existing table data is uncompressed. New data will be not be compressed.

Notes:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 V9.7 or later, or if the table is migrated using an online table move.
2. If a dictionary exists and the compression attribute is enabled but there currently isn't any data in the table, the `RESETDICTIONARY` operation will keep the existing dictionary. Rows which are smaller in size than the internal minimum record length and rows which do not demonstrate a savings in record length when an attempt is made to compress them are considered insufficient in this case.
3. If DATA CAPTURE CHANGES option is specified on the CREATE TABLE or ALTER TABLE statements, the current data dictionary is kept (referred to as the *historical compression dictionary*).

ALL DBPARTITIONNUMS

Specifies that operation is to be done on all database partitions specified in the db2nodes.cfg file. This is the default if a node clause is not specified.

EXCEPT
Specifies that operation is to be done on all database partitions specified in the db2nodes.cfg file, except those specified in the node list.

ON DATA PARTITION *partition-name*
Specifies the data partition for the reorganization.

This option is only for use with RECLAIM EXTENTS ONLY.

If the data partition name does not exist for the specified table, REORG fails and SQL0204N is returned.

ON DBPARTITIONNUM | ON DBPARTITIONNUMS
Perform operation on a set of database partitions.

db-partition-number1
Specifies a database partition number in the database partition list.

db-partition-number2
Specifies the second database partition number, so that all database partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

Usage notes

Restrictions:

- The REORG utility does not support the use of nicknames.
- The REORG TABLE command is not supported for declared temporary tables or created temporary tables.
- The REORG TABLE command cannot be used on views.
- Reorganization of a table is not compatible with range-clustered tables, because the range area of the table always remains clustered.
- REORG TABLE cannot be used on a partitioned table in a DMS table space while an online backup of ANY table space in which the table resides, including LOBs and indexes, is being performed.
- REORG TABLE cannot use an index that is based on an index extension.
- If a table is in reorg pending state, an inplace reorg is not allowed on the table.
- For partitioned tables:
  - The table must have an ACCESS_MODE in SYSCAT.TABLES of Full Access.
  - Reorganization skips data partitions that are in a restricted state due to an attach or detach operation. If the Table partitioning clause is specified, that partition must be fully accessible.
  - If an error occurs during table reorganization, some indexes or index partitions might be left invalid. The nonpartitioned indexes of the table will be marked invalid if the reorganization has reached or passed the replace phase for the first data partition. The index partitions for any data partition that has already reached or passed the replace phase will be marked invalid. Indexes will be rebuilt on the next access to the table or data partition.
  - If an error occurs during index reorganization when the ALLOW NONE access mode is used, some indexes on the table might be left invalid. For nonpartitioned RID indexes on the table, only the index that is being reorganized at the time of the failure will be left invalid. For MDC tables with nonpartitioned block indexes, one or more of the block indexes might be left

invalid if an error occurs. For partitioned indexes, only the index object on the data partition being reorganized will be left invalid. Any indexes marked invalid will be rebuilt on the next access to the table or data partition.

Information about the current progress of table reorganization is written to the history file for database activity. The history file contains a record for each reorganization event. To view this file, execute the LIST HISTORY command for the database that contains the table you are reorganizing.

You can also use table snapshots to monitor the progress of table reorganization. Table reorganization monitoring data is recorded regardless of the Database Monitor Table Switch setting.

If an error occurs, an SQLCA dump is written to the history file. For an in-place table reorganization, the status is recorded as PAUSED.

When an indexed table has been modified many times, the data in the indexes might become fragmented. If the table is clustered with respect to an index, the table and index can get out of cluster order. Both of these factors can adversely affect the performance of scans using the index, and can impact the effectiveness of index page prefetching. REORG INDEX or REORG INDEXES can be used to reorganize one or all of the indexes on a table. Index reorganization will remove any fragmentation and restore physical clustering to the leaf pages. Use the REORGCHK command to help determine if an index needs reorganizing. Be sure to complete all database operations and release all locks before invoking index reorganization. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

A classic table reorganization (offline reorganization) rebuilds the indexes during the last phase of the reorganization. However, the inplace table reorganization (online reorganization) does not rebuild the indexes. It is recommended that you issue a REORG INDEXES command after the completion of an inplace table reorganization. An inplace table reorganization is asynchronous, therefore care must be taken to ensure that the inplace table reorganization is complete before issuing the REORG INDEXES command. Issuing the REORG INDEXES command before the inplace table reorganization is complete, might cause the reorganization to fail (SQLCODE -2219).

Tables that have been modified so many times that data is fragmented and access performance is noticeably slow are candidates for the REORG TABLE command. You should also invoke this utility after altering the inline length of a structured type column in order to benefit from the altered inline length. Use the REORGCHK command to determine whether a table needs reorganizing. Be sure to complete all database operations and release all locks before invoking REORG TABLE. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK. After reorganizing a table, use RUNSTATS to update the table statistics, and REBIND to rebind the packages that use this table. The reorganize utility will implicitly close all the cursors.

If the table contains mixed row format because the table value compression has been activated or deactivated, an offline table reorganization can convert all the existing rows into the target row format.

If the table is distributed across several database partitions, and the table or index reorganization fails on any of the affected database partitions, only the failing database partitions will have the table or index reorganization rolled back.

If the reorganization is not successful, temporary files should not be deleted. The database manager uses these files to recover the database.

If the name of an index is specified, the database manager reorganizes the data according to the order in the index. To maximize performance, specify an index that is often used in SQL queries. If the name of an index is *not* specified, and if a clustering index exists, the data will be ordered according to the clustering index.

The PCTFREE value of a table determines the amount of free space designated per page. If the value has not been set, the utility will fill up as much space as possible on each page.

To complete a table space roll-forward recovery following a table reorganization, both regular and large table spaces must be enabled for roll-forward recovery.

If the table contains LOB columns that do not use the COMPACT option, the LOB DATA storage object can be significantly larger following table reorganization. This can be a result of the order in which the rows were reorganized, and the types of table spaces used (SMS or DMS).

Indexes over XML data may be recreated by the REORG INDEXES/TABLE command. For details, see "Recreation of indexes over XML data".

REORG INDEXES/TABLE

Reorganizes an index or a table

You can reorganize all indexes defined on a table by rebuilding the index data into unfragmented, physically contiguous pages. Alternatively, you have the option of reorganizing specific indexes on a range partitioned table.

If you specify the CLEANUP ONLY option of the index clause, cleanup is performed without rebuilding the indexes. This command cannot be used against indexes on declared temporary tables or created temporary tables (SQLSTATE 42995).

The table option reorganizes a table by reconstructing the rows to eliminate fragmented data, and by compacting information.

Scope

This command affects all database partitions in the database partition group.

Authorization

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *sqladm*
- CONTROL privilege on the table.

Required connection

Database

Command syntax

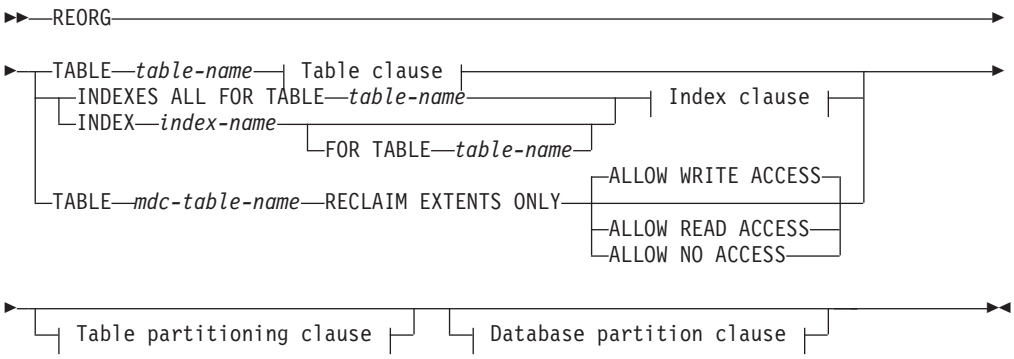
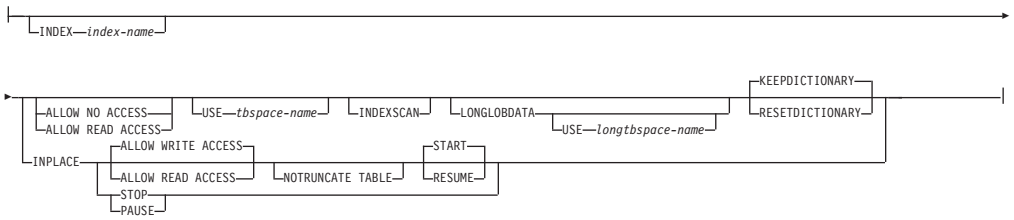


Table clause:



Index clause:

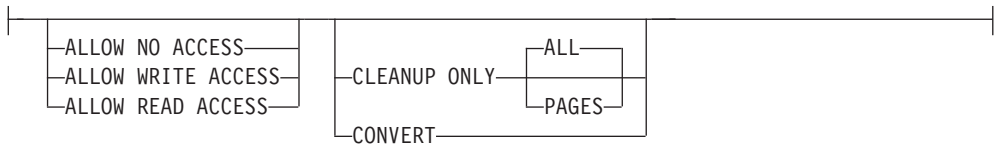
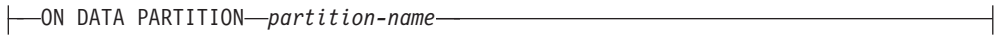
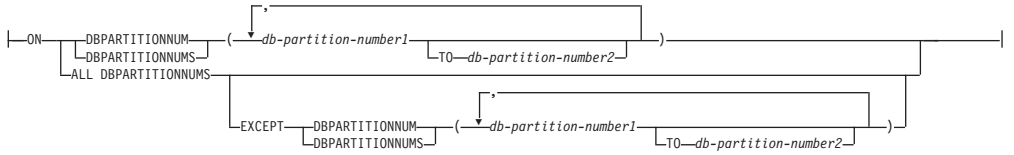


Table partitioning clause:



Database partition clause:



Command parameters

INDEXES ALL FOR TABLE *table-name*
Specifies the table whose indexes are to be reorganized. The table can be in a local or a remote database.

INDEX *index-name*
Specifies an individual index to be reorganized on a partitioned table. Reorganization of individual indexes are *only* supported for nonpartitioned indexes on a partitioned table. This parameter is not supported for block indexes.

FOR TABLE *table-name*
Specifies the table name location of the individual index being reorganized on a partitioned table. This parameter is optional, given that index names are unique across the database.

For REORG INDEXES when the ON DATA PARTITION option is specified, the access clause only applies to the named partition. Users can read from and write to the rest of the table while the indexes on the specified partition are being reorganized. This situation also applies to the default access levels.

ALLOW NO ACCESS
For REORG INDEX and REORG INDEXES, specifies that no other users can access the table or data partition while the indexes are being reorganized.

ALLOW READ ACCESS
For REORG INDEX and REORG INDEXES, specifies that other users can have read-only access to the table or data partition while the indexes are being reorganized. This access level is not supported for REORG INDEXES of a partitioned table unless the CLEANUP ONLY option or the ON DATA PARTITION option is specified.

ALLOW WRITE ACCESS
For REORG INDEX and REORG INDEXES, specifies that other users can read from and write to the table or data partition while the indexes are being reorganized.

This access level is not supported for multidimensional clustering (MDC) tables or extended indexes unless the CLEANUP ONLY option is specified. In addition, this access level is not supported for partitioned tables when the REORG INDEXES command is used unless the CLEANUP ONLY option or the ON DATA PARTITION option is specified.

When ACCESS mode is not specified, it is selected for the table or data partition in the following way:

Table 165. Default table access chosen based on the command, table type and additional parameters specified for the index clause:

Command	Table type	Table partitioning clause	Additional parameters specified for index clause	Default access mode
REORG INDEXES	Nonpartitioned table	Not applicable	Any	ALLOW READ ACCESS
REORG INDEX	Partitioned table	Not applicable	Any	ALLOW READ ACCESS
REORG INDEXES	Partitioned table	None	None specified	ALLOW NO ACCESS

Table 165. Default table access chosen based on the command, table type and additional parameters specified for the index clause: (continued)

Command	Table type	Table partitioning clause	Additional parameters specified for index clause	Default access mode
REORG INDEXES	Partitioned table	ON DATA PARTITION	None specified	ALLOW READ ACCESS
REORG INDEXES	Partitioned table	With or without the ON DATA PARTITION clause	CLEANUP ONLY specified	ALLOW READ ACCESS

CLEANUP ONLY
When CLEANUP ONLY is requested, a cleanup rather than a full reorganization will be done. The indexes will not be rebuilt and any pages freed up will be available for reuse by indexes defined on this table only.

The CLEANUP ONLY PAGES option will search for and free committed pseudo empty pages. A committed pseudo empty page is one where all the keys on the page are marked as deleted and all these deletions are known to be committed. The number of pseudo empty pages in an indexes can be determined by running RUNSTATS and looking at the NUM EMPTY LEAFS column in SYSCAT.INDEXES. The PAGES option will clean the NUM EMPTY LEAFS if they are determined to be committed.

The CLEANUP ONLY ALL option will free committed pseudo empty pages, as well as remove committed pseudo deleted keys from pages that are not pseudo empty. This option will also try to merge adjacent leaf pages if doing so will result in a merged leaf page that has at least PCTFREE free space on the merged leaf page, where PCTFREE is the percent free space defined for the index at index creation time. The default PCTFREE is ten percent. If two pages can be merged, one of the pages will be freed. The number of pseudo deleted keys in an index , excluding those on pseudo empty pages, can be determined by running RUNSTATS and then selecting the NUMRIDS DELETED from SYSCAT.INDEXES. The ALL option will clean the NUMRIDS DELETED and the NUM EMPTY LEAFS if they are determined to be committed.

ALL Specifies that indexes should be cleaned up by removing committed pseudo deleted keys and committed pseudo empty pages.

PAGES
Specifies that committed pseudo empty pages should be removed from the index tree. This will not clean up pseudo deleted keys on pages that are not pseudo empty. Since it is only checking the pseudo empty leaf pages, it is considerably faster than using the ALL option in most cases.

CONVERT
Converts type-1 indexes to type-2 index. If the index is already type 2, this option has no effect.

In Version 9.7, type-1 indexes are discontinued and all indexes that are created are type-2 indexes. As a result, the CONVERT option is deprecated.

All indexes created prior to Version 8 are type-1 indexes. Prior to Version 9.7, all indexes created by Version 8 and later are type-2 indexes, except when you create an index on a table that already has a type-1 index. In this case, the new index was also of type 1. This is no longer the case in Version 9.7 because all indexes created are type 2.

Use the ALLOW READ ACCESS or ALLOW WRITE ACCESS option to allow other transactions either read-only or read-write access to the table while the indexes are being reorganized. While ALLOW READ ACCESS and ALLOW WRITE ACCESS allow access to the table, during the period in which the reorganized copies of the indexes are made available, no access to the table is allowed.

TABLE *mdc-table-name* **RECLAIM EXTENTS ONLY**
Specifies the multidimensional clustering (MDC) table to reorganize to reclaim extents that are not being used. The name or alias in the form: *schema.table-name* can be used. The *schema* is the user name under which the table was created. If you omit the schema name, the default schema is assumed.

For REORG TABLE RECLAIM EXTENTS ONLY when the ON DATA PARTITION option is specified, the access clause only applies to the named partition. Users can read from and write to the rest of the table while the extents on the specified partition are being reclaimed. This situation also applies to the default access levels.

ALLOW NO ACCESS
For REORG TABLE RECLAIM EXTENTS ONLY, specifies that no other users can access the table while the extents are being reclaimed.

ALLOW READ ACCESS
For REORG TABLE RECLAIM EXTENTS ONLY, specifies that other users can have read-only access to the table while the extents are being reclaimed.

ALLOW WRITE ACCESS
For REORG TABLE RECLAIM EXTENTS ONLY, specifies that other users can read from and write to the table while the extents are being reclaimed.

TABLE *table-name*
Specifies the table to reorganize. The table can be in a local or a remote database. The name or alias in the form: *schema.table-name* can be used. The *schema* is the user name under which the table was created. If you omit the schema name, the default schema is assumed.

For typed tables, the specified table name must be the name of the hierarchy's root table.

You cannot specify an index for the reorganization of a multidimensional clustering (MDC) table. In place reorganization of tables cannot be used for MDC tables.

INDEX *index-name*
Specifies the index to use when reorganizing the table. If you do

not specify the fully qualified name in the form: *schema.index-name*, the default schema is assumed. The *schema* is the user name under which the index was created. The database manager uses the index to physically reorder the records in the table it is reorganizing.

For an in place table reorganization, if a clustering index is defined on the table and an index is specified, it must be clustering index. If the in place option is not specified, any index specified will be used. If you do not specify the name of an index, the records are reorganized without regard to order. If the table has a clustering index defined, however, and no index is specified, then the clustering index is used to cluster the table. You cannot specify an index if you are reorganizing an MDC table.

ALLOW NO ACCESS
Specifies that no other users can access the table while the table is being reorganized. When reorganizing a partitioned table with no Table partitioning clause, this value is the default.

ALLOW READ ACCESS
Allow only read access to the table during reorganization. This value is the default for a nonpartitioned table or for a partitioned table using the Table partitioning clause.

When the ON DATA PARTITION option is specified for a REORG TABLE of a range partitioned table, the access clause applies to the named partition only. If all indexes on the table are partitioned, users can read from and write to the rest of the table while the partition is being reorganized and its indexes rebuild. However, if any nonpartitioned indexes are defined for the table, access for the entire table is escalated to ALLOW NO ACCESS. In this situation, the nonpartitioned indexes are rebuilt along with the specified partition.

INPLACE
Reorganizes the table while permitting user access.

In place table reorganization is allowed only on nonpartitioned and non-MDC tables with type-2 indexes, but without extended indexes and with no indexes defined over XML columns in the table. In place table reorganization can only be performed on tables that are at least three pages in size.

In place table reorganization takes place asynchronously, and might not be effective immediately.

ALLOW READ ACCESS
Allow only read access to the table during reorganization.

ALLOW WRITE ACCESS
Allow write access to the table during reorganization. This is the default behavior.

NOTRUNCATE TABLE
Do not truncate the table after in place reorganization. During truncation, the table is S-locked.

START
Start the in place REORG processing. Because this is the default, this keyword is optional.

STOP Stop the in place REORG processing at its current point.

PAUSE
Suspend or pause in place REORG for the time being.

RESUME
Continue or resume a previously paused in place table reorganization. When an online reorganization is resumed and you want the same options as when the reorganization was paused, you must specify those options again while resuming.

USE *tbspace-name*
Specifies the name of a system temporary table space in which to store a temporary copy of the table being reorganized. If you do not provide a table space name, the database manager stores a working copy of the table in the table spaces that contain the table being reorganized.

For an 8KB, 16KB, or 32KB table object, if the page size of the system temporary table space that you specify does not match the page size of the table spaces in which the table data resides, the DB2 database product will try to find a temporary table space of the correct size of the LONG/LOB objects. Such a table space must exist for the reorganization to succeed.

When you have two temporary table spaces of the same page size, and you specify one of them in the USE clause, they will be used in a round robin fashion if there is an index in the table being reorganized. Say you have two table spaces, `tempspace1` and `tempspace2`, both of the same page size and you specify `tempspace1` in the REORG command with the USE option. When you perform REORG the first time, `tempspace1` is used. The second time, `tempspace2` is used. The third time, `tempspace1` is used and so on. To avoid this, you should drop one of the temporary table spaces.

For partitioned tables, the table space is used as temporary storage for the reorganization of data partitions in the table. Reorganization of the entire partitioned table reorganizes a single data partition at a time. The amount of space required is equal to the largest data partition in the table, and not the entire table.

If you do not supply a table space name for a partitioned table, the table space where each data partition is located is used for temporary storage of that data partition. There must be enough free space in each data partition's table space to hold a copy of the data partition.

INDEXSCAN
For a clustering REORG an index scan will be used to re-order table records. Reorganize table rows by accessing the table through an index. The default method is to scan the table and sort the result to reorganize the table, using temporary table spaces as necessary. Even though the index keys are in sort order, scanning and sorting is typically faster than fetching rows by first reading the row identifier from an index.

LONGLOBDATA
Long field and LOB data are to be reorganized.

This is not required even if the table contains long or LOB columns. The default is to avoid reorganizing these objects because

it is time consuming and does not improve clustering. However, running a reorganization with the LONGLOBDATA option on tables with XML columns will reclaim unused space and thereby reduce the size of the XML storage object.

This parameter is required when converting existing LOB data into inlined LOB data.

USE *longtbspace-name*
This is an optional parameter, which can be used to specify the name of a temporary table space to be used for rebuilding long data. If no temporary table space is specified for either the table object or for the long objects, the objects will be constructed in the table space they currently reside. If a temporary table space is specified for the table but this parameter is not specified, then the table space used for base reorg data will be used, unless the page sizes differ. In this situation, the DB2 database system will attempt to choose a temporary container of the appropriate page size to create the long objects in.

If USE *longtbspace-name* is specified, USE *tbspace-name* must also be specified. If it is not, the *longtbspace-name* argument is ignored.

KEEPDICTIONARY
If the `COMPRESS` attribute for the table is `YES` and the table has a compression dictionary then no new dictionary is built. All the rows processed during reorganization are subject to compression using the existing dictionary. If the `COMPRESS` attribute is `YES` and a compression dictionary doesn't exist for the table, a dictionary will only be created (and the table compressed) in this scenario if the table is of a certain size (approximately 1 to 2 MB) and sufficient data exists within this table. If, instead, you explicitly state REORG RESETDICTIONARY, then a dictionary is built as long as there is at least 1 row in the table. If the `COMPRESS` attribute for the table is `NO` and the table has a compression dictionary, then reorg processing will preserve the dictionary and all the rows in the newly reorganized table will be in noncompressed format. It is not possible to compress some data such as LOB data not stored in the base table row.

When the LONGLOBDATA option is not specified, only the table row data is reorganized. The following table describes the behavior of KEEPDICTIONARY syntax in REORG command when the LONGLOBDATA option is not specified.

Table 166. REORG KEEPDICTIONARY

Compress	Dictionary Exists	Result; outcome
Y	Y	Preserve dictionary; rows compressed.
Y	N	Build dictionary; rows compressed
N	Y	Preserve dictionary; all rows uncompressed
N	N	No effect; all rows uncompressed

The following table describes the behavior of KEEPDICTIONARY syntax in REORG command when the LONGLOBDATA option is specified.

Table 167. REORG KEEPDICTIONARY when LONGLOBDATA option is specified.

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Compression dictionary	Data compression
Y	Y	Y	Preserve dictionaries.	Existing data is compressed. New data will be compressed.
Y	Y	N	Preserve table row dictionary and create an XML storage object dictionary.	Existing data is compressed. New data will be compressed.
Y	N	Y	Create table row dictionary and preserve the XML dictionary.	Existing data is compressed. New data will be compressed.
Y	N	N	Create table row and XML dictionaries.	Existing data is compressed. New data will be compressed.
N	Y	Y	Preserve table row and XML dictionaries.	Table data is uncompressed. New data will be not be compressed.
N	Y	N	Preserve table row dictionary.	Table data is uncompressed. New data will be not be compressed.
N	N	Y	Preserve XML dictionary.	Table data is uncompressed. New data will be not be compressed.
N	N	N	No effect.	Table data is uncompressed. New data will be not be compressed.

Note:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 V9.7 or later, or if the table is migrated using the ONLINE_TABLE_MOVE stored procedure.

For any reinitialization or truncation of a table (such as for a replace operation), if the compress attribute for the table is NO, the dictionary is discarded if one exists. Conversely, if a dictionary exists and the compress attribute for the table is YES then a truncation will save the dictionary and not discard it. The dictionary is logged in its entirety for recovery purposes and for future support with data capture changes (that is, replication).

RESETDICTIONARY

If the COMPRESS attribute for the table is YES then a new row compression dictionary is built. All the rows processed during reorganization are subject to compression using this new dictionary. This dictionary replaces any previous dictionary. If the

COMPRESS attribute for the table is NO and the table does have an existing compression dictionary then reorg processing will remove the dictionary and all rows in the newly reorganized table will be in noncompressed format. It is not possible to compress some data such as LOB data not stored in the base table row.

If the LONGLOBDATA option is not specified, only the table row data is reorganized. The following table describes the behavior of RESETDICTIONARY syntax in REORG command when the LONGLOBDATA option is not specified.

Table 168. REORG RESETDICTIONARY

Compress	Dictionary Exists	Result; outcome
Y	Y	Build new dictionary*; rows compressed. If DATA CAPTURE CHANGES option is specified on the CREATE TABLE or ALTER TABLE statements, the current dictionary is kept (referred to as the *historical compression dictionary*).
Y	N	Build new dictionary; rows compressed
N	Y	Remove dictionary; all rows uncompressed. If the DATA CAPTURE NONE option is specified on the CREATE TABLE or ALTER TABLE statements, the *historical compression dictionary* is also removed for the specified table.
N	N	No effect; all rows uncompressed

* - If a dictionary exists and the compression attribute is enabled but there currently isn't any data in the table, the RESETDICTIONARY operation will keep the existing dictionary. Rows which are smaller in size than the internal minimum record length and rows which do not demonstrate a savings in record length when an attempt is made to compress them are considered "insufficient" in this case.

The following table describes the behavior of RESETDICTIONARY syntax in REORG command when the LONGLOBDATA option is specified.

Table 169. REORG RESETDICTIONARY when LONGLOBDATA option is specified.

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Data dictionary	Data compression
Y	Y	Y	Build dictionaries[2] [3].	Existing data is compressed. New data will be compressed.
Y	Y	N	Build new table row dictionary and create a new XML dictionary[3].	Existing data is compressed. New data will be compressed.
Y	N	Y	Create table row data dictionary and build a new XML dictionary.	Existing data is compressed. New data will be compressed.
Y	N	N	Create dictionaries.	Existing data is compressed. New data will be compressed.

Table 169. REORG RESETDICTIONARY when LONGLOBDATA option is specified. (continued)

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Data dictionary	Data compression
N	Y	Y	Remove dictionaries. Existing and new data is not compressed.	Existing table data is uncompressed. New data will be not be compressed.
N	Y	N	Remove table row dictionary. All data is uncompressed.	Existing table data is uncompressed. New data will be not be compressed.
N	N	Y	Remove XML storage object dictionary.	Existing table data is uncompressed. New data will be not be compressed.
N	N	N	No effect.	Existing table data is uncompressed. New data will be not be compressed.

Notes:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 V9.7 or later, or if the table is migrated using an online table move.
2. If a dictionary exists and the compression attribute is enabled but there currently isn't any data in the table, the `RESETDICTIONARY` operation will keep the existing dictionary. Rows which are smaller in size than the internal minimum record length and rows which do not demonstrate a savings in record length when an attempt is made to compress them are considered insufficient in this case.
3. If DATA CAPTURE CHANGES option is specified on the CREATE TABLE or ALTER TABLE statements, the current data dictionary is kept (referred to as the *historical compression dictionary*).

ALL DBPARTITIONNUMS
Specifies that operation is to be done on all database partitions specified in the db2nodes.cfg file. This is the default if a node clause is not specified.

EXCEPT
Specifies that operation is to be done on all database partitions specified in the db2nodes.cfg file, except those specified in the node list.

ON DATA PARTITION *partition-name*
Specifies the data partition for the reorganization.

This option is only for use with RECLAIM EXTENTS ONLY.

If the data partition name does not exist for the specified table, REORG fails and SQL0204N is returned.

ON DBPARTITIONNUM | ON DBPARTITIONNUMS
Perform operation on a set of database partitions.

db-partition-number1
Specifies a database partition number in the database partition list.

db-partition-number2
Specifies the second database partition number, so that all database partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

Usage notes

Restrictions:

- The REORG utility does not support the use of nicknames.
- The REORG TABLE command is not supported for declared temporary tables or created temporary tables.
- The REORG TABLE command cannot be used on views.
- Reorganization of a table is not compatible with range-clustered tables, because the range area of the table always remains clustered.
- REORG TABLE cannot be used on a partitioned table in a DMS table space while an online backup of ANY table space in which the table resides, including LOBs and indexes, is being performed.
- REORG TABLE cannot use an index that is based on an index extension.
- If a table is in reorg pending state, an inplace reorg is not allowed on the table.
- For partitioned tables:
  - The table must have an ACCESS_MODE in SYSCAT.TABLES of Full Access.
  - Reorganization skips data partitions that are in a restricted state due to an attach or detach operation. If the Table partitioning clause is specified, that partition must be fully accessible.
  - If an error occurs during table reorganization, some indexes or index partitions might be left invalid. The nonpartitioned indexes of the table will be marked invalid if the reorganization has reached or passed the replace phase for the first data partition. The index partitions for any data partition that has already reached or passed the replace phase will be marked invalid. Indexes will be rebuilt on the next access to the table or data partition.
  - If an error occurs during index reorganization when the ALLOW NONE access mode is used, some indexes on the table might be left invalid. For nonpartitioned RID indexes on the table, only the index that is being reorganized at the time of the failure will be left invalid. For MDC tables with nonpartitioned block indexes, one or more of the block indexes might be left invalid if an error occurs. For partitioned indexes, only the index object on the data partition being reorganized will be left invalid. Any indexes marked invalid will be rebuilt on the next access to the table or data partition.

Information about the current progress of table reorganization is written to the history file for database activity. The history file contains a record for each reorganization event. To view this file, execute the LIST HISTORY command for the database that contains the table you are reorganizing.

You can also use table snapshots to monitor the progress of table reorganization. Table reorganization monitoring data is recorded regardless of the Database Monitor Table Switch setting.

If an error occurs, an SQLCA dump is written to the history file. For an in-place table reorganization, the status is recorded as PAUSED.

When an indexed table has been modified many times, the data in the indexes might become fragmented. If the table is clustered with respect to an index, the table and index can get out of cluster order. Both of these factors can adversely affect the performance of scans using the index, and can impact the effectiveness of index page prefetching. REORG INDEX or REORG INDEXES can be used to reorganize one or all of the indexes on a table. Index reorganization will remove any fragmentation and restore physical clustering to the leaf pages. Use the REORGCHK command to help determine if an index needs reorganizing. Be sure to complete all database operations and release all locks before invoking index reorganization. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

A classic table reorganization (offline reorganization) rebuilds the indexes during the last phase of the reorganization. However, the inplace table reorganization (online reorganization) does not rebuild the indexes. It is recommended that you issue a REORG INDEXES command after the completion of an inplace table reorganization. An inplace table reorganization is asynchronous, therefore care must be taken to ensure that the inplace table reorganization is complete before issuing the REORG INDEXES command. Issuing the REORG INDEXES command before the inplace table reorganization is complete, might cause the reorganization to fail (SQLCODE -2219).

Tables that have been modified so many times that data is fragmented and access performance is noticeably slow are candidates for the REORG TABLE command. You should also invoke this utility after altering the inline length of a structured type column in order to benefit from the altered inline length. Use the REORGCHK command to determine whether a table needs reorganizing. Be sure to complete all database operations and release all locks before invoking REORG TABLE. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK. After reorganizing a table, use RUNSTATS to update the table statistics, and REBIND to rebind the packages that use this table. The reorganize utility will implicitly close all the cursors.

If the table contains mixed row format because the table value compression has been activated or deactivated, an offline table reorganization can convert all the existing rows into the target row format.

If the table is distributed across several database partitions, and the table or index reorganization fails on any of the affected database partitions, only the failing database partitions will have the table or index reorganization rolled back.

If the reorganization is not successful, temporary files should not be deleted. The database manager uses these files to recover the database.

If the name of an index is specified, the database manager reorganizes the data according to the order in the index. To maximize performance, specify an index that is often used in SQL queries. If the name of an index is *not* specified, and if a clustering index exists, the data will be ordered according to the clustering index.

The PCTFREE value of a table determines the amount of free space designated per page. If the value has not been set, the utility will fill up as much space as possible on each page.

To complete a table space roll-forward recovery following a table reorganization, both regular and large table spaces must be enabled for roll-forward recovery.

If the table contains LOB columns that do not use the COMPACT option, the LOB DATA storage object can be significantly larger following table reorganization. This can be a result of the order in which the rows were reorganized, and the types of table spaces used (SMS or DMS).

Indexes over XML data may be recreated by the REORG INDEXES/TABLE command. For details, see "Recreation of indexes over XML data".

RESTART DATABASE

Restarts a database that has been abnormally terminated and left in an inconsistent state. At the successful completion of RESTART DATABASE, the application remains connected to the database if the user has CONNECT privilege.

Scope

This command affects only the node on which it is executed.

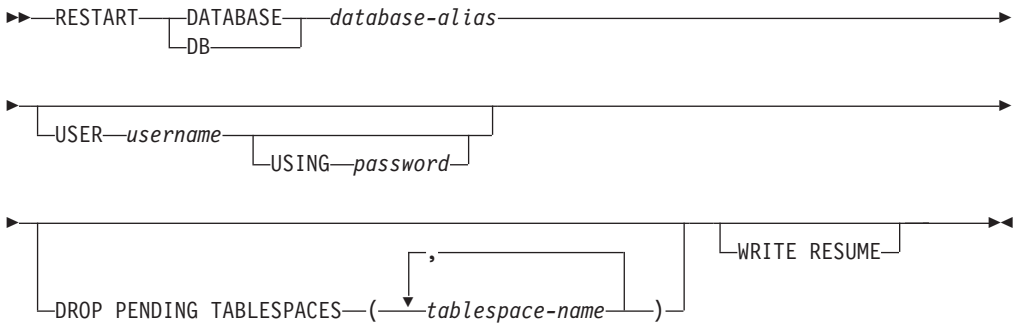
Authorization

None

Required connection

This command establishes a database connection.

Command syntax



Command parameters

DATABASE *database-alias*
Identifies the database to restart.

USER *username*
Identifies the user name under which the database is to be restarted.

USING *password*
The password used to authenticate *username*. If the password is omitted, the user is prompted to enter it.

DROP PENDING TABLESPACES *tablespace-name*
Specifies that the database restart operation is to be successfully completed even if table space container problems are encountered.

If a problem occurs with a container for a specified table space during the restart process, the corresponding table space will not be available (it will be in drop-pending state) after the restart operation. If a table space is in the drop-pending state, the only possible action is to drop the table space.

In the case of circular logging, a troubled table space will cause a restart failure. A list of troubled table space names can found in the administration notification log if a restart database operation fails because of container problems. If there is only one system temporary table space in the database, and it is in drop pending state, a new system temporary table space must be created immediately following a successful database restart operation.

WRITE RESUME
Allows you to force a database restart on databases that failed while I/O writes were suspended. Before performing crash recovery, this option will resume I/O writes by removing the SUSPEND_WRITE state from every table space in the database.

The WRITE RESUME option can also be used in the case where the connection used to suspend I/O writes is currently hung and all subsequent connection attempts are also hanging. When used in this circumstance, RESTART DATABASE will resume I/O writes to the database without performing crash recovery. RESTART DATABASE with the WRITE RESUME option will only perform crash recovery when you use it after a database crash. The WRITE RESUME parameter can only be applied to the primary database, not to mirrored databases.

Usage notes

Execute this command if an attempt to connect to a database returns an error message, indicating that the database must be restarted. This action occurs only if the previous session with this database terminated abnormally (due to power failure, for example).

On a partitioned database system, in order to resolve the indoubt transactions, the RESTART DATABASE command should be issued on all nodes, as in the example below:

```
db2_all "db2 restart database database-alias"
```

If the database is only restarted on a single node within an MPP system, a message might be returned on a subsequent database query indicating that the database needs to be restarted. This occurs because the database partition on a node on which the query depends must also be restarted. Restarting the database on all nodes solves the problem.

RESTORE DATABASE

The RESTORE DATABASE command recreates a damaged or corrupted database that has been backed up using the DB2 backup utility. The restored database is in the same state that it was in when the backup copy was made. This utility can also overwrite a database with a different image or restore the backup copy to a new database.

For information on the restore operations supported by DB2 database systems between different operating systems and hardware platforms, see "Backup and restore operations between different operating systems and hardware platforms" in the *Data Recovery and High Availability Guide and Reference*.

The restore utility can also be used to restore backup images in DB2 Version 9.7 that were backed up on DB2 Universal Database Version 8, DB2 Version 9.1, or DB2 Version 9.5. If a database upgrade is required, it will be invoked automatically at the end of the restore operation.

If, at the time of the backup operation, the database was enabled for rollforward recovery, the database can be brought to its previous state by invoking the rollforward utility after successful completion of a restore operation.

This utility can also restore a table space level backup.

Incremental images and images only capturing differences from the time of the previous capture (called a "delta image") cannot be restored when there is a difference in operating systems or word size (32-bit or 64-bit).

Following a successful restore operation from one environment to a different environment, no incremental or delta backups are allowed until a non-incremental backup is taken. (This is not a limitation following a restore operation within the same environment.)

Even with a successful restore operation from one environment to a different environment, there are some considerations: packages must be rebound before use (using the BIND command, the REBIND command, or the db2rbind utility); SQL procedures must be dropped and recreated; and all external libraries must be rebuilt on the new platform. (These are not considerations when restoring to the same environment.)

A restore operation run over an existing database and existing containers reuses the same containers and tablespace map.

A restore operation run against a new database reacquires all containers and rebuilds an optimized tablespace map. A restore operation run over an existing database with one or more missing containers also reacquires all containers and rebuilds an optimized tablespace map.

Scope

This command only affects the node on which it is executed.

Authorization

To restore to an existing database, one of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

To restore to a new database, one of the following:

- *sysadm*
- *sysctrl*

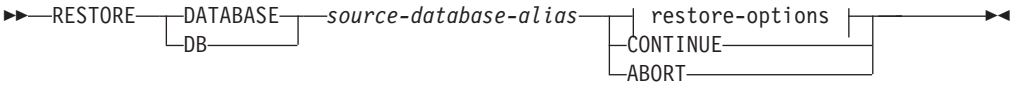
Required connection

The required connection will vary based on the type of restore action:

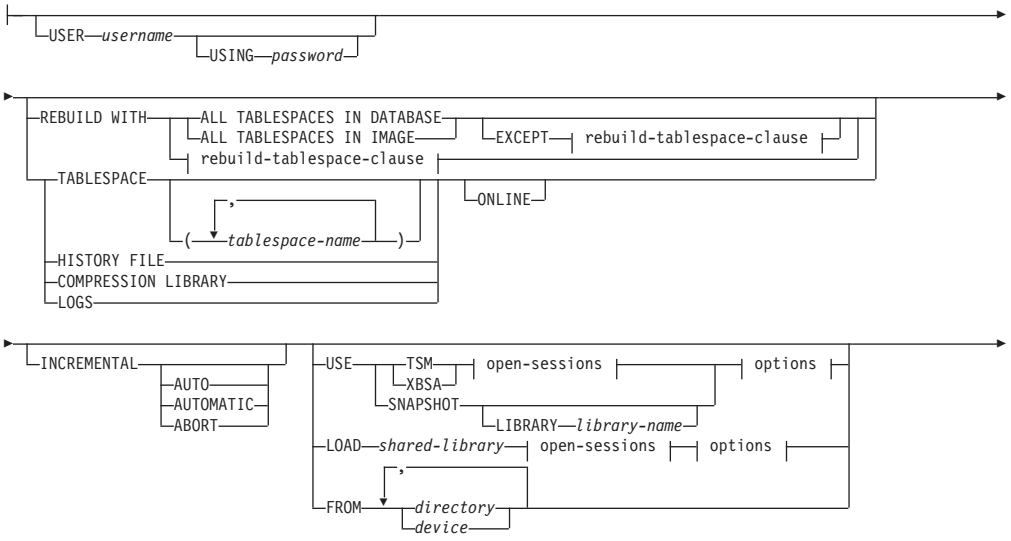
- You require a database connection, to restore to an existing database. This command automatically establishes an exclusive connection to the specified database.
- You require an instance and a database connection, to restore to a new database. The instance attachment is required to create the database.

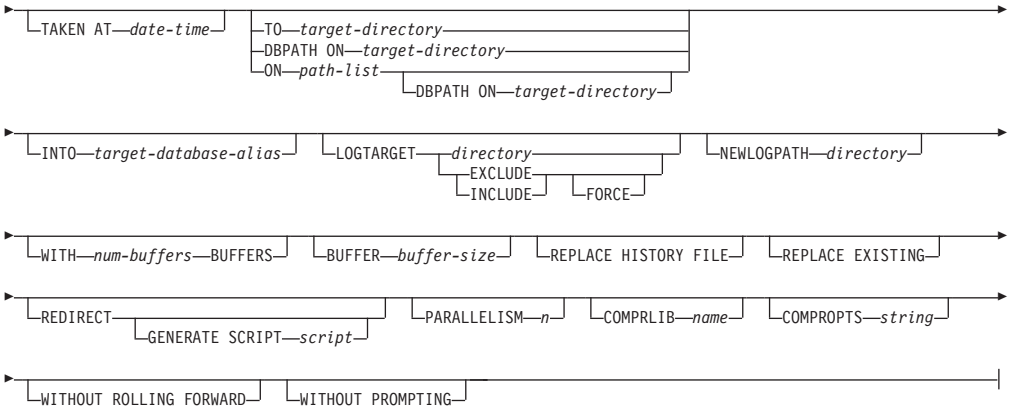
  To restore to a new database at an instance different from the current instance, it is necessary to first attach to the instance where the new database will reside. The new instance can be local or remote. The current instance is defined by the value of the DB2INSTANCE environment variable.
- For snapshot restore, *instance* and *database* connections are required.

Command syntax

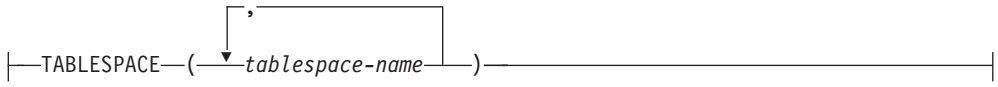


restore-options:





rebuild-tablespace-clause:



open-sessions:

options:

Command parameters

DATABASE *source-database-alias*
Alias of the source database from which the backup was taken.

CONTINUE
Specifies that the containers have been redefined, and that the final step in a redirected restore operation should be performed.

ABORT
This parameter:

- Stops a redirected restore operation. This is useful when an error has occurred that requires one or more steps to be repeated. After RESTORE DATABASE with the ABORT option has been issued, each step of a redirected restore operation must be repeated, including RESTORE DATABASE with the REDIRECT option.
- Terminates an incremental restore operation before completion.

USER *username*
Identifies the user name under which the database is to be restored.

USING *password*
The password used to authenticate the user name. If the password is omitted, the user is prompted to enter it.

REBUILD WITH ALL TABLESPACES IN DATABASE
Restores the database with all the table spaces known to the database at the time of the image being restored. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN DATABASE EXCEPT *rebuild-tablespace-clause*
Restores the database with all the table spaces known to the database at the time of the image being restored except for those specified in the list. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN IMAGE
Restores the database with only the table spaces in the image being restored. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN IMAGE EXCEPT *rebuild-tablespace-clause*
Restores the database with only the table spaces in the image being restored except for those specified in the list. This restore overwrites a database if it already exists.

REBUILD WITH *rebuild-tablespace-clause*
Restores the database with only the list of table spaces specified. This restore overwrites a database if it already exists.

TABLESPACE *tablespace-name*
A list of names used to specify the table spaces that are to be restored.

ONLINE
This keyword, applicable only when performing a table space-level restore operation, is specified to allow a backup image to be restored online. This means that other agents can connect to the database while the backup image is being restored, and that the data in other table spaces will be available while the specified table spaces are being restored.

HISTORY FILE
This keyword is specified to restore only the history file from the backup image.

COMPRESSION LIBRARY
This keyword is specified to restore only the compression library from the backup image. If the object exists in the backup image, it will be restored into the database directory. If the object does not exist in the backup image, the restore operation will fail.

LOGS This keyword is specified to restore only the set of log files contained in the backup image. If the backup image does not contain any log files, the restore operation will fail. If this option is specified, the LOGTARGET option must also be specified.

INCREMENTAL
Without additional parameters, INCREMENTAL specifies a manual cumulative restore operation. During manual restore the user must issue each restore command manually for each image involved in the restore. Do so according to the following order: last, first, second, third and so on up to and including the last image.

INCREMENTAL AUTOMATIC/AUTO
Specifies an automatic cumulative restore operation.

INCREMENTAL ABORT
Specifies abortion of an in-progress manual cumulative restore operation.

USE

TSM Specifies that the database is to be restored from output managed by Tivoli Storage Manager.

XBSA Specifies that the XBSA interface is to be used. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.

SNAPSHOT
Specifies that the data is to be restored from a snapshot backup.

You cannot use the SNAPSHOT parameter with any of the following parameters:

- INCREMENTAL
- TO
- ON
- DBPATH ON
- INTO
- NEWLOGPATH
- WITH *num-buffers* BUFFERS
- BUFFER
- REDIRECT
- REPLACE HISTORY FILE
- COMPRESSION LIBRARY
- PARALLELISM
- COMPRLIB
- OPEN *num-sessions* SESSIONS
- HISTORY FILE
- LOGS

Also, you cannot use the SNAPSHOT parameter with any restore operation that involves a table space list, which includes the REBUILD WITH option.

The default behavior when restoring data from a snapshot backup image will be a FULL DATABASE OFFLINE restore of all paths that make up the database including all containers, local volume directory, database path (DBPATH), primary log and mirror log paths of the most recent snapshot backup if no timestamp is provided (INCLUDE LOGS is the default for all snapshot backups unless EXCLUDE LOGS is explicitly stated). If a timestamp is provided, then that snapshot backup image will be restored.

LIBRARY *library-name*
Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage® SAN Volume Controller
- IBM Enterprise Storage Server® Model 800
- IBM System Storage™ DS6000™
- IBM System Storage DS8000®
- IBM System Storage N Series
- NetApp V-series

- NetApp FAS

If you have other storage hardware, and a DB2 ACS API driver for that storage hardware, you can use the LIBRARY parameter to specify the DB2 ACS API driver.

The value of the LIBRARY parameter is a fully-qualified library file name.

OPTIONS

"options-string"
Specifies options to be used for the restore operation. The string will be passed to DB2 ACS API driver exactly as it was entered, without the double quotation marks. You cannot use the **VENDOROPT** database configuration parameter to specify vendor-specific options for snapshot restore operations. You must use the OPTIONS parameter of the restore utilities instead.

@file-name
Specifies that the options to be used for the restore operation are contained in a file located on the DB2 server. The string will be passed to the vendor support library. The file must be a fully qualified file name.

OPEN *num-sessions* **SESSIONS**
Specifies the number of I/O sessions that are to be used with TSM or the vendor product.

FROM *directory/device*
The fully qualified path name of the directory or device on which the backup image resides. If USE TSM, FROM, and LOAD are omitted, the default value is the current working directory of the client machine. This target directory or device must exist on the target server/instance.

If several items are specified, and the last item is a tape device, the user is prompted for another tape. Valid response options are:

c Continue. Continue using the device that generated the warning message (for example, continue when a new tape has been mounted).

d Device terminate. Stop using *only* the device that generated the warning message (for example, terminate when there are no more tapes).

t Terminate. Abort the restore operation after the user has failed to perform some action requested by the utility.

LOAD *shared-library*
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. The name can contain a full path. If the full path is not given, the value defaults to the path on which the user exit program resides.

TAKEN AT® *date-time*
The time stamp of the database backup image. The time stamp is displayed after successful completion of a backup operation, and is part of the path name for the backup image. It is specified in the form *yyyymmddhhmmss*. A partial time stamp can also be specified. For example, if two different backup images with time stamps 20021001010101 and 20021002010101 exist, specifying `20021002` causes the image with time

stamp 20021002010101 to be used. If a value for this parameter is not specified, there must be only one backup image on the source media.

TO *target-directory*
This parameter states the target database directory. This parameter is ignored if the utility is restoring to an existing database. The drive and directory that you specify must be local. If the backup image contains a database that is enabled for automatic storage then only the database directory changes, the storage paths associated with the database do not change.

DBPATH ON *target-directory*
This parameter states the target database directory. This parameter is ignored if the utility is restoring to an existing database. The drive and directory that you specify must be local. If the backup image contains a database that is enabled for automatic storage and the ON parameter is not specified then this parameter is synonymous with the TO parameter and only the database directory changes, the storage paths associated with the database do not change.

ON *path-list*
This parameter redefines the storage paths associated with an automatic storage database. Using this parameter with a database that is not enabled for automatic storage results in an error (SQL20321N). The existing storage paths as defined within the backup image are no longer used and automatic storage table spaces are automatically redirected to the new paths. If this parameter is not specified for an automatic storage database then the storage paths remain as they are defined within the backup image.

One or more paths can be specified, each separated by a comma. Each path must have an absolute path name and it must exist locally. If the database does not already exist on disk and the DBPATH ON parameter is not specified then the first path is used as the target database directory.

For a multi-partition database the ON *path-list* option can only be specified on the catalog partition. The catalog partition must be restored before any other partitions are restored when the ON option is used. The restore of the catalog-partition with new storage paths will place all non-catalog nodes in a RESTORE_PENDING state. The non-catalog nodes can then be restored in parallel without specifying the ON clause in the restore command.

In general, the same storage paths must be used for each partition in a multi-partition database and they must all exist prior to executing the RESTORE DATABASE command. One exception to this is where database partition expressions are used within the storage path. Doing this allows the database partition number to be reflected in the storage path such that the resulting path name is different on each partition.

You use the argument " $N" ([blank]$N) to indicate a database partition expression. A database partition expression can be used anywhere in the storage path, and multiple database partition expressions can be specified. Terminate the database partition expression with a space character; whatever follows the space is appended to the storage path after the database partition expression is evaluated. If there is no space character in the storage path after the database partition expression, it is assumed that the rest of the string is part of the expression. The argument can only be used in one of the following forms:

Table 170. . Operators are evaluated from left to right. % represents the modulus operator. The database partition number in the examples is assumed to be 10.

Syntax	Example	Value
[blank]$N	″ $N″	10
[blank]$N+[number]	″ $N+100″	110
[blank]$N%[number]	″ $N%5″	0
[blank]$N+[number]%[number]	″ $N+1%5″	1
[blank]$N%[number]+[number]	″ $N%4+2″	4
[a] % is modulus.		

INTO *target-database-alias*
The target database alias. If the target database does not exist, it is created.

When you restore a database backup to an existing database, the restored database inherits the alias and database name of the existing database. When you restore a database backup to a nonexistent database, the new database is created with the alias and database name that you specify. This new database name must be unique on the system where you restore it.

LOGTARGET *directory*
Non-snapshot restores:

The absolute path name of an existing directory on the database server, to be used as the target directory for extracting log files from a backup image. If this option is specified, any log files contained within the backup image will be extracted into the target directory. If this option is not specified, log files contained within a backup image will not be extracted. To extract only the log files from the backup image, specify the LOGS option.

Snapshot restores:

INCLUDE
Restore log directory volumes from the snapshot image. If this option is specified and the backup image contains log directories, then they will be restored. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If existing log directories on disk conflict with the log directories in the backup image, then an error will be returned.

EXCLUDE
Do not restore log directory volumes. If this option is specified, then no log directories will be restored from the backup image. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If a path belonging to the database is restored and a log directory will implicitly be restored because of this, thus causing a log directory to be overwritten, an error will be returned.

FORCE
Allow existing log directories in the current database to be overwritten and replaced when restoring the snapshot image. Without this option, existing log directories and log files on disk which conflict with log directories in the snapshot image will cause the restore to fail. Use this option to indicate that the restore can overwrite and replace those existing log directories.

Note: Use this option with caution, and always ensure that you have backed up and archived all logs that might be required for recovery.

Note: If LOGTARGET is not specified non-snapshot restores, then the default LOGTARGET directory is LOGTARGET EXCLUDE.

NEWLOGPATH *directory*
The absolute pathname of a directory that will be used for active log files after the restore operation. This parameter has the same function as the **newlogpath** database configuration parameter, except that its effect is limited to the restore operation in which it is specified. The parameter can be used when the log path in the backup image is not suitable for use after the restore operation; for example, when the path is no longer valid, or is being used by a different database.

WITH *num-buffers* **BUFFERS**
The number of buffers to be used. The DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value. A larger number of buffers can be used to improve performance when multiple sources are being read from, or if the value of PARALLELISM has been increased.

BUFFER *buffer-size*
The size, in pages, of the buffer used for the restore operation. The DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value. The minimum value for this parameter is 8 pages.

The restore buffer size must be a positive integer multiple of the backup buffer size specified during the backup operation. If an incorrect buffer size is specified, the buffers are allocated to be of the smallest acceptable size.

REPLACE HISTORY FILE
Specifies that the restore operation should replace the history file on disk with the history file from the backup image.

REPLACE EXISTING
If a database with the same alias as the target database alias already exists, this parameter specifies that the restore utility is to replace the existing database with the restored database. This is useful for scripts that invoke the restore utility, because the command line processor will not prompt the user to verify deletion of an existing database. If the WITHOUT PROMPTING parameter is specified, it is not necessary to specify REPLACE EXISTING, but in this case, the operation will fail if events occur that normally require user intervention.

REDIRECT
Specifies a redirected restore operation. To complete a redirected restore operation, this command should be followed by one or more SET TABLESPACE CONTAINERS commands, and then by a RESTORE DATABASE command with the CONTINUE option. All commands associated with a single redirected restore operation must be invoked from the same window or CLP session.

GENERATE SCRIPT *script*
Creates a redirect restore script with the specified file name. The script name can be relative or absolute and the script will be generated on the client side. If the file cannot be created on the client side, an error message

(SQL9304N) will be returned. If the file already exists, it will be overwritten. Please see the examples below for further usage information.

WITHOUT ROLLING FORWARD
Specifies that the database is not to be put in rollforward pending state after it has been successfully restored.

If, following a successful restore operation, the database is in rollforward pending state, the ROLLFORWARD command must be invoked before the database can be used again.

If this option is specified when restoring from an online backup image, error SQL2537N will be returned.

If backup image is of a recoverable database then WITHOUT ROLLING FORWARD cannot be specified with REBUILD option.

PARALLELISM *n*
Specifies the number of buffer manipulators that are to be created during the restore operation. The DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value.

COMPRLIB *name*
Indicates the name of the library to be used to perform the decompression (e.g., `db2compr.dll` for Windows; `libdb2compr.so` for Linux/UNIX systems). The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library cannot be loaded, the restore operation will fail.

COMPROPTS *string*
Describes a block of binary data that is passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so any issues of byte reversal or code page conversion are handled by the decompression library. If the first character of the data block is "@", the remainder of the data is interpreted by the DB2 database system as the name of a file residing on the server. The DB2 database system will then replace the contents of *string* with the contents of this file and pass the new value to the initialization routine instead. The maximum length for the string is 1 024 bytes.

WITHOUT PROMPTING
Specifies that the restore operation is to run unattended. Actions that normally require user intervention will return an error message. When using a removable media device, such as tape or diskette, the user is prompted when the device ends, even if this option is specified.

Examples

1. In the following example, the database WSDB is defined on all 4 database partitions, numbered 0 through 3. The path /dev3/backup is accessible from all database partitions. The following offline backup images are available from /dev3/backup:

```
wsdb.0.db2inst1.NODE0000.CATN0000.20020331234149.001
wsdb.0.db2inst1.NODE0001.CATN0000.20020331234427.001
wsdb.0.db2inst1.NODE0002.CATN0000.20020331234828.001
wsdb.0.db2inst1.NODE0003.CATN0000.20020331235235.001
```

To restore the catalog partition first, then all other database partitions of the WSDB database from the /dev3/backup directory, issue the following commands from one of the database partitions:

```
db2_all '<<+0< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234149
  INTO wsdb REPLACE EXISTING'
db2_all '<<+1< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234427
  INTO wsdb REPLACE EXISTING'
db2_all '<<+2< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234828
  INTO wsdb REPLACE EXISTING'
db2_all '<<+3< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331235235
  INTO wsdb REPLACE EXISTING'
```

The db2_all utility issues the restore command to each specified database partition. When performing a restore using db2_all, you should always specify REPLACE EXISTING and/or WITHOUT PROMPTING. Otherwise, if there is prompting, the operation will look like it is hanging. This is because db2_all does not support user prompting.

2. Following is a typical redirected restore scenario for a database whose alias is MYDB:
   a. Issue a RESTORE DATABASE command with the REDIRECT option.

      ```
      restore db mydb replace existing redirect
      ```

      After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

      ```
      restore db mydb abort
      ```

   b. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example:

      ```
      set tablespace containers for 5 using
         (file 'f:\ts3con1' 20000, file 'f:\ts3con2' 20000)
      ```

      To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.

   c. After successful completion of steps 1 and 2, issue:

      ```
      restore db mydb continue
      ```

      This is the final step of the redirected restore operation.

   d. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.
3. Following is a sample weekly incremental backup strategy for a recoverable database. It includes a weekly full database backup operation, a daily non-cumulative (delta) backup operation, and a mid-week cumulative (incremental) backup operation:

   ```
   (Sun) backup db mydb use tsm
   (Mon) backup db mydb online incremental delta use tsm
   (Tue) backup db mydb online incremental delta use tsm
   (Wed) backup db mydb online incremental use tsm
   (Thu) backup db mydb online incremental delta use tsm
   (Fri) backup db mydb online incremental delta use tsm
   (Sat) backup db mydb online incremental use tsm
   ```

   For an automatic database restore of the images created on Friday morning, issue:

```
restore db mydb incremental automatic taken at (Fri)
```

For a manual database restore of the images created on Friday morning, issue:

```
restore db mydb incremental taken at (Fri)
restore db mydb incremental taken at (Sun)
restore db mydb incremental taken at (Wed)
restore db mydb incremental taken at (Thu)
restore db mydb incremental taken at (Fri)
```

4. To produce a backup image, which includes logs, for transportation to a remote site:

   ```
   backup db sample online to /dev3/backup include logs
   ```

   To restore that backup image, supply a LOGTARGET path and specify this path during ROLLFORWARD:

   ```
   restore db sample from /dev3/backup logtarget /dev3/logs
   rollforward db sample to end of logs and stop overflow log path /dev3/logs
   ```

5. To retrieve only the log files from a backup image that includes logs:

   ```
   restore db sample logs from /dev3/backup logtarget /dev3/logs
   ```

6. The USE TSM OPTIONS keywords can be used to specify the TSM information to use for the restore operation. On Windows platforms, omit the -fromowner option.
   - Specifying a delimited string:

     ```
     restore db sample use TSM options '"-fromnode=bar -fromowner=dmcinnis"'
     ```

   - Specifying a fully qualified file:

     ```
     restore db sample use TSM options @/u/dmcinnis/myoptions.txt
     ```

     The file `myoptions.txt` contains the following information: -fromnode=bar -fromowner=dmcinnis

7. The following is a simple restore of a multi-partition automatic storage enabled database with new storage paths. The database was originally created with one storage path, `/myPath0`:
   - On the catalog partition issue: `restore db mydb on /myPath1,/myPath2`
   - On all non-catalog partitions issue: `restore db mydb`

8. A script output of the following command on a non-auto storage database:

   ```
   restore db sample from /home/jseifert/backups taken at 20050301100417 redirect
   generate script SAMPLE_NODE0000.clp
   ```

   would look like this:

```
-- ******************************************************************************
-- ** automatically created redirect restore script
-- ******************************************************************************
UPDATE COMMAND OPTIONS USING S ON Z ON SAMPLE_NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM  0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
-- ******************************************************************************
-- ** initialize redirected restore
-- ******************************************************************************
RESTORE DATABASE SAMPLE
-- USER  '<username>'
-- USING '<password>'
FROM '/home/jseifert/backups'
TAKEN AT 20050301100417
-- DBPATH ON '<target-directory>'
INTO SAMPLE
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00001/SQLOGDIR/'
-- WITH <num-buff> BUFFERS
```

```
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
-- ******************************************************************************
-- ** tablespace definition
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = SYSCATSPACE
-- **   Tablespace ID                         = 0
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 5572
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 0
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0000.0'
);
-- ******************************************************************************
-- ** Tablespace name                         = TEMPSPACE1
-- **   Tablespace ID                         = 1
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = System Temporary data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 0
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 1
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0001.0'
);
-- ******************************************************************************
-- ** Tablespace name                         = USERSPACE1
-- **   Tablespace ID                         = 2
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 1
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 2
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0002.0'
);
-- ******************************************************************************
-- ** Tablespace name                         = DMS
-- **   Tablespace ID                         = 3
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Auto-resize enabled                   = No
-- **   Total number of pages                 = 2000
-- **   Number of usable pages                = 1960
```

```
-- **   High water mark (pages)                 = 96
-- ********************************************************************************
SET TABLESPACE CONTAINERS FOR 3
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   /tmp/dms1                                                        1000
, FILE   /tmp/dms2                                                        1000
);
-- ********************************************************************************
-- ** Tablespace name                         = RAW
-- **   Tablespace ID                         = 4
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Auto-resize enabled                   = No
-- **   Total number of pages                 = 2000
-- **   Number of usable pages                = 1960
-- **   High water mark (pages)               = 96
-- ********************************************************************************
SET TABLESPACE CONTAINERS FOR 4
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  DEVICE '/dev/hdb1'                                          1000
, DEVICE '/dev/hdb2'                                          1000
);
-- ********************************************************************************
-- ** start redirect restore
-- ********************************************************************************
RESTORE DATABASE SAMPLE CONTINUE;
-- ********************************************************************************
-- ** end of file
-- ********************************************************************************
```

9. A script output of the following command on an automatic storage database:

   ```
   restore db test from /home/jseifert/backups taken at 20050304090733 redirect
   generate script TEST_NODE0000.clp
   ```

   would look like this:

```
-- ********************************************************************************
-- ** automatically created redirect restore script
-- ********************************************************************************
UPDATE COMMAND OPTIONS USING S ON Z ON TEST_NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM  0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
-- ********************************************************************************
-- ** initialize redirected restore
-- ********************************************************************************
RESTORE DATABASE TEST
-- USER  '<username>'
-- USING '<password>'
FROM '/home/jseifert/backups'
TAKEN AT 20050304090733
ON '/home/jseifert'
-- DBPATH ON <target-directory>
INTO TEST
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00002/SQLOGDIR/'
-- WITH <num-buff> BUFFERS
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
```

```
;
-- ******************************************************************************
-- ** tablespace definition
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = SYSCATSPACE
-- **   Tablespace ID                         = 0
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 4
-- **   Using automatic storage               = Yes
-- **   Auto-resize enabled                   = Yes
-- **   Total number of pages                 = 6144
-- **   Number of usable pages                = 6140
-- **   High water mark (pages)               = 5968
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = TEMPSPACE1
-- **   Tablespace ID                         = 1
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = System Temporary data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = Yes
-- **   Total number of pages                 = 0
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = USERSPACE1
-- **   Tablespace ID                         = 2
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = Yes
-- **   Auto-resize enabled                   = Yes
-- **   Total number of pages                 = 256
-- **   Number of usable pages                = 224
-- **   High water mark (pages)               = 96
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = DMS
-- **   Tablespace ID                         = 3
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Auto-resize enabled                   = No
-- **   Total number of pages                 = 2000
-- **   Number of usable pages                = 1960
-- **   High water mark (pages)               = 96
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 3
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   '/tmp/dms1'                                        1000
, FILE   '/tmp/dms2'                                        1000
);
-- ******************************************************************************
-- ** Tablespace name                         = RAW
-- **   Tablespace ID                         = 4
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
```

```
-- **   Auto-resize enabled                    = No
-- **   Total number of pages                  = 2000
-- **   Number of usable pages                 = 1960
-- **   High water mark (pages)                = 96
-- ********************************************************************************
SET TABLESPACE CONTAINERS FOR 4
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  DEVICE '/dev/hdb1'                                    1000
, DEVICE '/dev/hdb2'                                    1000
);
-- ********************************************************************************
-- ** start redirect restore
-- ********************************************************************************
RESTORE DATABASE TEST CONTINUE;
-- ********************************************************************************
-- ** end of file
-- ********************************************************************************
```

10. The following are examples of the RESTORE DB command using the SNAPSHOT option:

    Restore log directory volumes from the snapshot image and do not prompt.

    ```
    db2 restore db sample use snapshot LOGTARGET INCLUDE without prompting
    ```

    Do not restore log directory volumes and do not prompt.

    ```
    db2 restore db sample use snapshot LOGTARGET EXCLUDE without prompting
    ```

    Do not restore log directory volumes and do not prompt. When LOGTARGET is not specified, then the default is LOGTARGET EXCLUDE.

    ```
    db2 restore db sample use snapshot without prompting
    ```

    Allow existing log directories in the current database to be overwritten and replaced when restoring the snapshot image containing conflicting log directories, without prompting.

    ```
    db2 restore db sample use snapshot LOGTARGET EXCLUDE FORCE without prompting
    ```

    Allow existing log directories in the current database to be overwritten and replaced when restoring the snapshot image containing conflicting log directories, without prompting.

    ```
    db2 restore db sample use snapshot LOGTARGET INCLUDE FORCE without prompting
    ```

Usage notes

- A RESTORE DATABASE command of the form `db2 restore db <name>` will perform a full database restore with a database image and will perform a table space restore operation of the table spaces found in a table space image. A RESTORE DATABASE command of the form `db2 restore db <name> tablespace` performs a table space restore of the table spaces found in the image. In addition, if a list of table spaces is provided with such a command, the explicitly listed table spaces are restored.
- Following the restore operation of an online backup, you must perform a roll-forward recovery.
- If a backup image is compressed, the DB2 database system detects this and automatically decompresses the data before restoring it. If a library is specified on the db2Restore API, it is used for decompressing the data. Otherwise, a check is made to see if a library is stored in the backup image and if the library exists, it is used. Finally, if there is not library stored in the backup image, the data cannot be decompressed and the restore operation fails.
- If the compression library is to be restored from a backup image (either explicitly by specifying the COMPRESSION LIBRARY option or implicitly by performing a normal restore of a compressed backup), the restore operation must be done on the same platform and operating system that the backup was

taken on. If the platform the backup was taken on is not the same as the platform that the restore is being done on, the restore operation will fail, even if DB2 normally supports cross-platform restores involving the two systems.

- A backed up SMS tablespace can only be restored into a SMS tablespace. You cannot restore it into a DMS tablespace, or vice versa.
- To restore log files from the backup image that contains them, the LOGTARGET option must be specified, providing the fully qualified and valid path that exists on the DB2 server. If those conditions are satisfied, the restore utility will write the log files from the image to the target path. If a LOGTARGET is specified during a restore of a backup image that does not include logs, the restore operation will return an error before attempting to restore any table space data. A restore operation will also fail with an error if an invalid, or read-only, LOGTARGET path is specified.
- If any log files exist in the LOGTARGET path at the time the RESTORE DATABASE command is issued, a warning prompt will be returned to the user. This warning will not be returned if WITHOUT PROMPTING is specified.
- During a restore operation where a LOGTARGET is specified, if any log file cannot be extracted, the restore operation will fail and return an error. If any of the log files being extracted from the backup image have the same name as an existing file in the LOGTARGET path, the restore operation will fail and an error will be returned. The restore database utility will not overwrite existing log files in the LOGTARGET directory.
- You can also restore only the saved log set from a backup image. To indicate that only the log files are to be restored, specify the LOGS option in addition to the LOGTARGET path. Specifying the LOGS option without a LOGTARGET path will result in an error. If any problem occurs while restoring log files in this mode of operation, the restore operation will terminate immediately and an error will be returned.
- During an automatic incremental restore operation, only the log files included in the target image of the restore operation will be retrieved from the backup image. Any log files included in intermediate images referenced during the incremental restore process will not be extracted from those intermediate backup images. During a manual incremental restore operation, the LOGTARGET path should only be specified with the final restore command to be issued.
- Offline full database backups as well as offline incremental database backups can be restored to a later database version, whereas online backups cannot. For multi-partition databases, the catalog partition must first be restored individually, followed by the remaining database partitions (in parallel or serial). However, the implicit database upgrade done by the restore operation can fail. In a multi-partition database it can fail on one or more database partitions. In this case, you can follow the RESTORE DATABASE command with a single UPGRADE DATABASE command issued from the catalog partition to upgrade the database successfully.

Snapshot restore

Like a traditional (non-snapshot) restore, the default behavior when restoring a snapshot backup image will be to NOT restore the log directories —LOGTARGET EXCLUDE.

If the DB2 manager detects that any log directory's group ID is shared among any of the other paths to be restored, then an error is returned. In this case, LOGTARGET INCLUDE or LOGTARGET INCLUDE FORCE must be specified, as the log directories must be part of the restore.

The DB2 manager will make all efforts to save existing log directories (primary, mirror and overflow) before the restore of the paths from the backup image takes place.

If you wish the log directories to be restored and the DB2 manager detects that the pre-existing log directories on disk conflict with the log directories in the backup image, then the DB2 manager will report an error. In such a case, if you have specified LOGTARGET INCLUDE FORCE, then this error will be suppressed and the log directories from the image will be restored, deleting whatever existed beforehand.

There is a special case in which the LOGTARGET EXCLUDE option is specified and a log directory path resides under the database directory (i.e., /NODExxxx/SQLxxxxx/SQLOGDIR/). In this case, a restore would still overwrite the log directory as the database path, and all of the contents beneath it, would be restored. If the DB2 manager detects this scenario and log files exist in this log directory, then an error will be reported. If you specify LOGTARGET EXCLUDE FORCE, then this error will be suppressed and those log directories from the backup image will overwrite the conflicting log directories on disk.

ROLLFORWARD DATABASE

Recovers a database by applying transactions recorded in the database log files. Invoked after a database or a table space backup image has been restored, or if any table spaces have been taken offline by the database due to a media error. The database must be recoverable (that is, the **logarchmeth1** or **logarchmeth2** database configuration parameters must be set to a value other than `OFF`) before the database can be recovered with rollforward recovery.

Scope

In a partitioned database environment, this command can only be invoked from the catalog partition. A database or table space rollforward operation to a specified point in time affects all database partitions that are listed in the `db2nodes.cfg` file. A database or table space rollforward operation to the end of logs affects the database partitions that are specified. If no database partitions are specified, it affects all database partitions that are listed in the `db2nodes.cfg` file; if rollforward recovery is not needed on a particular partition, that partition is ignored.

For partitioned tables, you are also required to roll forward related table spaces to the same point in time. This applies to table spaces containing data partitions of a table. If a single table space contains a portion of a partitioned table, rolling forward to the end of the logs is still allowed.

It is not possible to roll forward through log files created on a previous DB2 release version. This is an important consideration when upgrading to a new DB2 release version.

Authorization

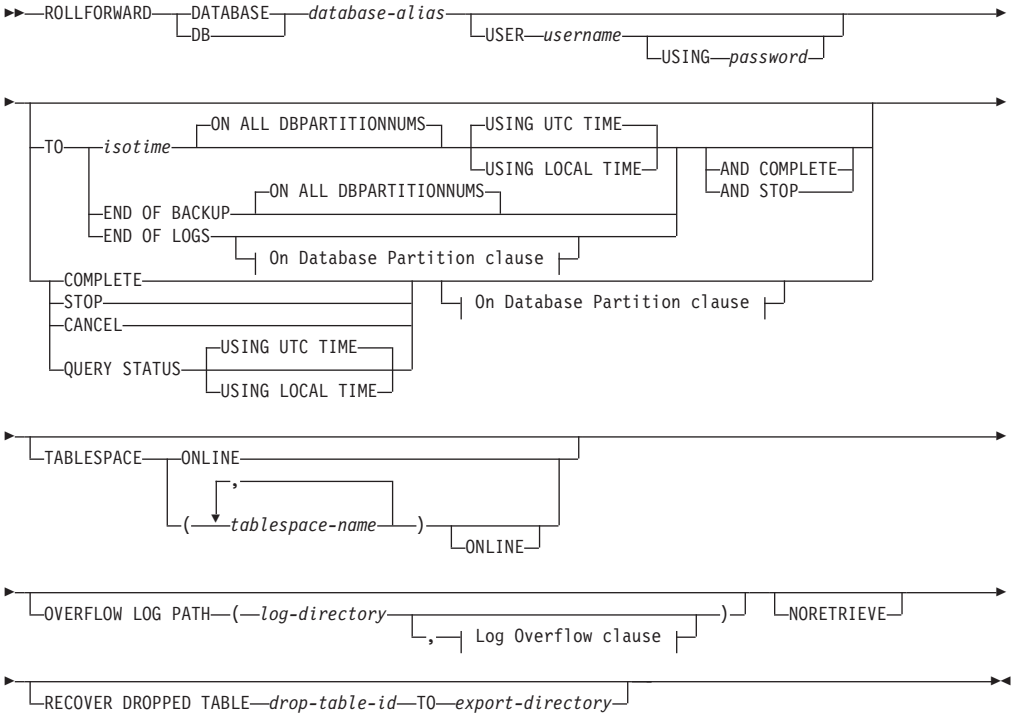
One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

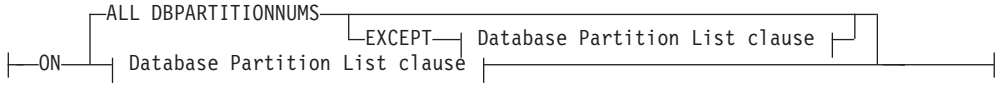
Required connection

None. This command establishes a database connection.

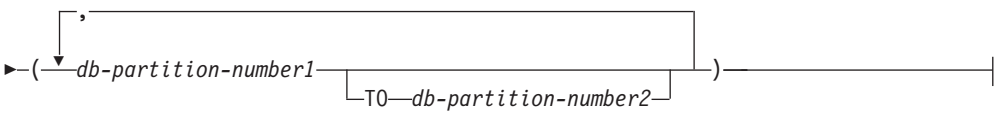
Command syntax



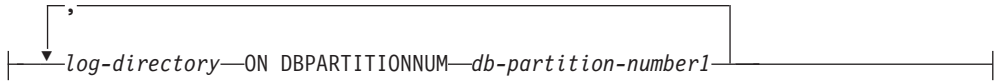
On Database Partition clause:



Database Partition List clause:



Log Overflow clause:



Command parameters

DATABASE *database-alias*
The alias of the database that is to be rollforward recovered.

USER *username*
The user name under which the database is to be rollforward recovered.

USING *password*
The password used to authenticate the user name. If the password is omitted, you will be prompted to enter it.

TO

isotime The point in time to which all committed transactions are to be rolled forward (including the transaction committed precisely at that time, as well as all transactions committed previously).

This value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss* (year, month, day, hour, minutes, seconds), expressed in Coordinated Universal Time (UTC, formerly known as GMT). UTC helps to avoid having the same time stamp associated with different logs (because of a change in time associated with daylight savings time, for example). The time stamp in a backup image is based on the local time at which the backup operation started. The CURRENT TIMEZONE special register specifies the difference between UTC and local time at the application server. The difference is represented by a time duration (a decimal number in which the first two digits represent the number of hours, the next two digits represent the number of minutes, and the last two digits represent the number of seconds). Subtracting CURRENT TIMEZONE from a local time converts that local time to UTC.

USING UTC TIME
Allows you to rollforward to a point in time that is specified as UTC time. This is the default option.

USING LOCAL TIME
Allows you to rollforward to a point in time that is the server's local time rather than UTC time.

Note:

1. If you specify a local time for rollforward, all messages returned to you will also be in local time. All times are converted on the server, and in partitioned database environments, on the catalog database partition.
2. The timestamp string is converted to UTC on the server, so the time is local to the server's time zone, not the client's. If the client is in one time zone and the server in another, the server's local time should be used. This is different from the local time option from the Control Center, which is local to the client.
3. If the timestamp string is close to the time change of the clock due to daylight savings, it is important to know if the stop time is before or after the clock change, and specify it correctly.
4. Subsequent ROLLFORWARD commands that cannot specify the USING LOCAL TIME clause will have all messages returned to you in local time if this option is specified.

5. It is important to choose the USING LOCAL TIME or the USING UTC TIME (formerly known as GMT time) correctly. If not specified, the default is USING UTC TIME. Any mistake in the selection may cause rollforward to reach a different point in time than expected and truncate the logs after that point in time. Mistaking a local timestamp as a UTC timestamp may cause the required logs to be truncated undesirably and prevent further rollforwards to a point later than the mistaken time.

END OF LOGS
Specifies that all committed transactions from all online archive log files listed in the database configuration parameter **logpath** are to be applied.

END OF BACKUP
Specifies that all partitions in the partitioned database should be rolled forward to the *minimum recovery time*. See Examples section below for an example.

ALL DBPARTITIONNUMS | ON ALL DBPARTITIONNUMS
Specifies that transactions are to be rolled forward on all database partitions specified in the `db2nodes.cfg` file. This is the default if a database partition clause is not specified.

EXCEPT
Specifies that transactions are to be rolled forward on all database partitions specified in the `db2nodes.cfg` file, except those specified in the database partition list.

ON DBPARTITIONNUM | ON DBPARTITIONNUMS
Roll the database forward on a set of database partitions.

db-partition-number1
Specifies a database partition number in the database partition list.

db-partition-number2
Specifies the second database partition number, so that all database partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

COMPLETE | STOP
Stops the rolling forward of log records, and completes the rollforward recovery process by rolling back any incomplete transactions and turning off the rollforward pending state of the database. This allows access to the database or table spaces that are being rolled forward. These keywords are equivalent; specify one or the other, but not both. The keyword AND permits specification of multiple operations at once; for example, `db2 rollforward db sample to end of logs and complete`. When rolling table spaces forward to a point in time, the table spaces are placed in backup pending state.

CANCEL
Cancels the rollforward recovery operation. This puts the database or one or more table spaces on all database partitions on which forward recovery has been started in restore pending state:

- If a *database* rollforward operation is not in progress (that is, the database is in rollforward pending state), this option puts the database in restore pending state.

- If a *table space* rollforward operation is not in progress (that is, the table spaces are in rollforward pending state), a table space list must be specified. All table spaces in the list are put in restore pending state.
- If a table space rollforward operation *is* in progress (that is, at least one table space is in rollforward in progress state), all table spaces that are in rollforward in progress state are put in restore pending state. If a table space list is specified, it must include all table spaces that are in rollforward in progress state. All table spaces on the list are put in restore pending state.
- If rolling forward to a point in time, any table space name that is passed in is ignored, and all table spaces that are in rollforward in progress state are put in restore pending state.
- If rolling forward to the end of the logs with a table space list, only the table spaces listed are put in restore pending state.

This option cannot be used to cancel a rollforward operation *that is actually running*. It can only be used to cancel a rollforward operation that is in progress but not actually running at the time. A rollforward operation can be in progress but not running if:

- It terminated abnormally.
- The STOP option was not specified.
- An error caused it to fail. Some errors, such as rolling forward through a non-recoverable load operation, can put a table space into restore pending state.

Use this option with caution, and only if the rollforward operation that is in progress cannot be completed because some of the table spaces have been put in rollforward pending state or in restore pending state. When in doubt, use the LIST TABLESPACES command to identify the table spaces that are in rollforward in progress state, or in rollforward pending state.

QUERY STATUS

Lists the log files that the database manager has rolled forward, the next archive file required, and the time stamp (in UTC) of the last committed transaction since rollforward processing began. In a partitioned database environment, this status information is returned for each database partition. The information returned contains the following fields:

Database partition number

Rollforward status

Status can be: database or table space rollforward pending, database or table space rollforward in progress, database or table space rollforward processing STOP, or not pending.

Next log file to be read

A string containing the name of the next required log file. In a partitioned database environment, use this information if the rollforward utility fails with a return code indicating that a log file is missing or that a log information mismatch has occurred.

Log files processed

A string containing the names of processed log files that are no longer needed for recovery, and that can be removed from the directory. If, for example, the oldest uncommitted transaction starts in log file *x*, the range of obsolete log files will not include *x*; the range ends at *x* - 1. This field is not updated in case of a table space rollforward recovery operation.

Last committed transaction
A string containing a time stamp in ISO format (*yyyy-mm-dd-hh.mm.ss*) suffixed by either "UTC" or "Local" (see USING LOCAL TIME). This time stamp marks the last transaction committed after the completion of rollforward recovery. The time stamp applies to the database. For table space rollforward recovery, it is the time stamp of the last transaction committed to the database.

QUERY STATUS is the default value if the TO, STOP, COMPLETE, or CANCEL clauses are omitted. If TO, STOP, or COMPLETE was specified, status information is displayed if the command has completed successfully. If individual table spaces are specified, they are ignored; the status request does not apply only to specified table spaces.

TABLESPACE
This keyword is specified for table space-level rollforward recovery.

tablespace-name
Mandatory for table space-level rollforward recovery to a point in time. Allows a subset of table spaces to be specified for rollforward recovery to the end of the logs. In a partitioned database environment, each table space in the list does not have to exist at each database partition that is rolling forward. If it *does* exist, it must be in the correct state.

For partitioned tables, point in time roll-forward of a table space containing any piece of a partitioned table must also roll-forward all of the other table spaces in which that table resides to the same point in time. The table spaces containing the index partitions are included in the list of pieces of a partitioned table. Roll-forward to the end of the logs for a single table space containing a piece of a partitioned table is still allowed.

If a partitioned table has any attached or detached data partitions, then PIT rollforward must include all table spaces for these data partitions as well. To determine if a partitioned table has any attached, detached, or dropped data partitions, query the Status field of the SYSDATAPARTITIONS catalog table.

Because a partitioned table can reside in multiple table spaces, it will generally be necessary to roll forward multiple table spaces. Data that is recovered via dropped table recovery is written to the export directory specified in the ROLLFORWARD DATABASE command. It is possible to roll forward all table spaces in one command, or do repeated roll forward operations for subsets of the table spaces involved. If the ROLLFORWARD DATABASE command is done for one or a few table spaces, then all data from the table that resided in those table spaces will be recovered. A warning will be written to the notify log if the ROLLFORWARD DATABASE command did not specify the full set of the table spaces necessary to recover all the data for the table. Allowing rollforward of a subset of the table spaces makes it easier to deal with cases where there is more data to be recovered than can fit into a single export directory.

ONLINE
This keyword is specified to allow table space-level rollforward recovery to be done online. This means that other agents are allowed to connect while rollforward recovery is in progress.

OVERFLOW LOG PATH *log-directory*
Specifies an alternate log path to be searched for archived logs during recovery. Use this parameter if log files were moved to a location other

than that specified by the **logpath** database configuration parameter. In a partitioned database environment, this is the (fully qualified) default overflow log path *for all database partitions*. A relative overflow log path can be specified for single-partition databases. The OVERFLOW LOG PATH command parameter will overwrite the value (if any) of the database configuration parameter **OVERFLOWLOGPATH**.

log-directory **ON DBPARTITIONNUM**
In a partitioned database environment, allows a different log path to override the default overflow log path for a specific database partition.

NORETRIEVE
Allows you to control which log files are to be rolled forward on the standby machine by allowing you to disable the retrieval of archived logs. The benefits of this are:

- By controlling the logfiles to be rolled forward, you can ensure that the standby machine is X hours behind the production machine, to avoid affecting both the systems.
- If the standby system does not have access to archive (eg. if TSM is the archive, it only allows the original machine to retrieve the files)
- It might also be possible that while the production system is archiving a file, the standby system is retrieving the same file, and it might then get an incomplete log file. Noretrieve would solve this problem.

RECOVER DROPPED TABLE *drop-table-id*
Recovers a dropped table during the rollforward operation. The table ID can be obtained using the LIST HISTORY command, in the Backup ID column of the output listing. For partitioned tables, the drop-table-id identifies the table as a whole, so that all data partitions of the table can be recovered in a single roll-forward command.

TO *export-directory*
Specifies a directory to which files containing the table data are to be written. The directory must be accessible to all database partitions.

Examples

Example 1

The ROLLFORWARD DATABASE command permits specification of multiple operations at once, each being separated with the keyword AND. For example, to roll forward to the end of logs, and complete, the separate commands:

```
db2 rollforward db sample to end of logs
db2 rollforward db sample complete
```

can be combined as follows:

```
db2 rollforward db sample to end of logs and complete
```

Although the two are equivalent, it is recommended that such operations be done in two steps. It is important to verify that the rollforward operation has progressed as expected, before stopping it and possibly missing logs. This is especially important if a bad log is found during rollforward recovery, and the bad log is interpreted to mean the “end of logs”. In such cases, an undamaged backup copy of that log could be used to continue the rollforward operation through more logs. However if the rollforward AND STOP option is used, and the rollforward encounters an error, the error will be returned to you. In this case, the only way to

force the rollforward to stop and come online despite the error (that is, to come online at that point in the logs before the error) is to issue the ROLLFORWARD STOP command.

Example 2

Roll forward to the end of the logs (two table spaces have been restored):

```
db2 rollforward db sample to end of logs
db2 rollforward db sample to end of logs and stop
```

These two statements are equivalent. Neither AND STOP or AND COMPLETE is needed for table space rollforward recovery to the end of the logs. Table space names are not required. If not specified, all table spaces requiring rollforward recovery will be included. If only a subset of these table spaces is to be rolled forward, their names must be specified.

Example 3

After three table spaces have been restored, roll one forward to the end of the logs, and the other two to a point in time, both to be done online:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online

db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   tablespace(TBS2, TBS3) online
```

Two rollforward operations cannot be run concurrently. The second command can only be invoked after the first rollforward operation completes successfully.

Example 4

After restoring the database, roll forward to a point in time, using OVERFLOW LOG PATH to specify the directory where the user exit saves archived logs:

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   overflow log path (/logs)
```

Example 5 (partitioned database environments)

There are three database partitions: 0, 1, and 2. Table space TBS1 is defined on all database partitions, and table space TBS2 is defined on database partitions 0 and 2. After restoring the database on database partition 1, and TBS1 on database partitions 0 and 2, roll the database forward on database partition 1:

```
db2 rollforward db sample to end of logs and stop
```

This returns warning SQL1271 (“Database is recovered but one or more table spaces are off-line on database partition(s) 0 and 2.”).

```
db2 rollforward db sample to end of logs
```

This rolls TBS1 forward on database partitions 0 and 2. The clause TABLESPACE(TBS1) is optional in this case.

Example 6 (partitioned database environments)

After restoring table space TBS1 on database partitions 0 and 2 only, roll TBS1 forward on database partitions 0 and 2:

```
db2 rollforward db sample to end of logs
```

Database partition 1 is ignored.

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1. Reports SQL4906N.

```
db2 rollforward db sample to end of logs on dbpartitionnums (0, 2)
   tablespace(TBS1)
```

This completes successfully.

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1; all pieces must be rolled forward together. With table space rollforward to a point in time, the database partition clause is not accepted. The rollforward operation must take place on all the database partitions on which the table space resides.

After restoring TBS1 on database partition 1:

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   tablespace(TBS1)
```

This completes successfully.

Example 7 (partitioned database environment)

After restoring a table space on all database partitions, roll forward to point in time 2, but do not specify `AND STOP`. The rollforward operation is still in progress. Cancel and roll forward to point in time 1:

```
db2 rollforward db sample to pit2 tablespace(TBS1)
db2 rollforward db sample cancel tablespace(TBS1)

** restore TBS1 on all database partitions **

db2 rollforward db sample to pit1 tablespace(TBS1)
db2 rollforward db sample stop tablespace(TBS1)
```

Example 8 (partitioned database environments)

Rollforward recover a table space that resides on eight database partitions (3 to 10) listed in the `db2nodes.cfg` file:

```
db2 rollforward database dwtest to end of logs tablespace (tssprodt)
```

This operation to the end of logs (not point in time) completes successfully. The database partitions on which the table space resides do not have to be specified. The utility defaults to the `db2nodes.cfg` file.

Example 9 (partitioned database environment)

Rollforward recover six small table spaces that reside on a single-partition database partition group (on database partition 6):

```
db2 rollforward database dwtest to end of logs on dbpartitionnum (6)
   tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

This operation to the end of logs (not point in time) completes successfully.

Example 10 (partitioned database environment)

You can use the TO END OF BACKUP clause with the ROLLFORWARD command to roll forward all partitions in a partitioned database to the minimum recovery time. The minimum recovery time is the earliest point in time during a rollforward when a database is consistent (when the objects listed in the database catalogs match the objects that physically exist on disk). Manually determining the correct point in time to which to roll forward a database is difficult, particularly for a partitioned database. The END OF BACKUP option makes it easy.

```
db2 rollforward db sample to end of backup and complete
```

Usage notes

If restoring from an image that was created during an online backup operation, the specified point in time for the rollforward operation must be later than the time at which the online backup operation completed. If the rollforward operation is stopped before it passes this point, the database is left in rollforward pending state. If a table space is in the process of being rolled forward, it is left in rollforward in progress state.

If one or more table spaces is being rolled forward to a point in time, the rollforward operation must continue at least to the minimum recovery time, which is the last update to the system catalogs for this table space or its tables. The minimum recovery time (in Coordinated Universal Time, or UTC) for a table space can be retrieved using the LIST TABLESPACES SHOW DETAIL command.

Rolling databases forward might require a load recovery using tape devices. If prompted for another tape, you can respond with one of the following:

c Continue. Continue using the device that generated the warning message (for example, when a new tape has been mounted)

d Device terminate. Stop using the device that generated the warning message (for example, when there are no more tapes)

t Terminate. Take all affected table spaces offline, but continue rollforward processing.

If the rollforward utility cannot find the next log that it needs, the log name is returned in the SQLCA, and rollforward recovery stops. If no more logs are available, use the STOP option to terminate rollforward recovery. Incomplete transactions are rolled back to ensure that the database or table space is left in a consistent state.

Note: Rolling forward through a redistribute operation cannot restore the database content since log records are not recorded for data redistribution. See the "REDISTRIBUTE DATABASE PARTITION GROUP command".

Compatibilities

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.
- The keyword NODES can be substituted for DBPARTITIONNUMS.
- Point in time rollforward is not supported with pre-V9.1 clients due to V9.1 support for partitioned tables.

SET RUNTIME DEGREE

Sets the maximum run time degree of intra-partition parallelism for SQL statements for specified active applications.

Scope

This command affects all database partitions that are listed in the `$HOME/sqllib/db2nodes.cfg` file.

Authorization

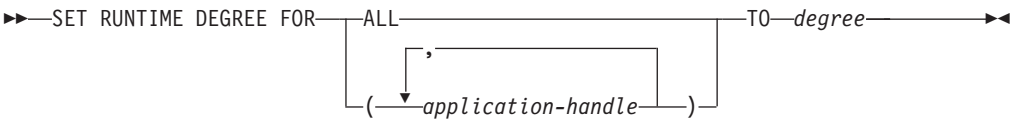
One of the following:

- *sysadm*
- *sysctrl*

Required connection

Instance. To change the maximum run time degree of intra-partition parallelism on a remote server, it is first necessary to attach to that server. If no attachment exists, the SET RUNTIME DEGREE command fails.

Command syntax



Command parameters

FOR

ALL The specified degree will apply to all applications.

application-handle
Specifies the agent to which the new degree applies. List the values using the LIST APPLICATIONS command.

TO *degree*
The maximum run time degree of intra-partition parallelism.

Examples

The following example sets the maximum run time degree of parallelism for two users, with *application-handle* values of 41408 and 55458, to 4:

```
db2 SET RUNTIME DEGREE FOR ( 41408, 55458 ) TO 4
```

Usage notes

This command provides a mechanism to modify the maximum degree of parallelism for active applications. It can be used to override the value that was determined at SQL statement compilation time.

The run time degree of intra-partition parallelism specifies the maximum number of parallel operations that will be used when the statement is executed. The degree

of intra-partition 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 run time degree of intra-partition parallelism for an active application can be specified using the SET RUNTIME DEGREE command. The *max_querydegree* database manager configuration parameter specifies the maximum run time degree for any SQL statement executing on this instance of the database manager.

The actual run time degree will be the lowest of:

- the **max_querydegree** configuration parameter
- the application run time degree
- the SQL statement compilation degree.

SET WRITE

The SET WRITE command allows a user to suspend I/O writes or to resume I/O writes for a database. Typical use of this command is for splitting a mirrored database. This type of mirroring is achieved through a disk storage system.

This new state, SUSPEND_WRITE, is visible from the Snapshot Monitor. All table spaces must be in a NORMAL state for the command to execute successfully. If any one table space is in a state other than NORMAL, the command will fail.

Scope

This command only affects the database partition on which it is executed.

Authorization

This command only affect the node on which it is executed. The authorization of this command requires the issuer to have one of the following privileges:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required Connection

Database

Command Syntax

Command Parameters

SUSPEND

Suspending I/O writes will put all table spaces into a new state SUSPEND_WRITE state. Writes to the logs are also suspended by this command. All database operations, apart from online backup and restore, should function normally while database writes are suspended. However, some operations can wait while attempting to flush dirty pages from the buffer pool or log buffers to the logs. These operations will resume normally once the database writes are resumed.

RESUME
Resuming I/O writes will remove the SUSPEND_WRITE state from all of the table spaces and make the table spaces available for update.

Usage notes

It is suggested that I/O writes be resumed from the same connection from which they were suspended. Ensuring that this connection is available to resume I/O writes involves not performing any operations from this connection until database writes are resumed. Otherwise, some operations can wait for I/O writes to be resumed if dirty pages must be flushed from the buffer pool or from log buffers to the logs. Furthermore, subsequent connection attempts might hang if they require flushing dirty pages from the buffer pool to disk. Subsequent connections will complete successfully once database I/O resumes. If your connection attempts are hanging, and it has become impossible to resume I/O from the connection that you used to suspend I/O, then you will have to run the RESTART DATABASE command with the WRITE RESUME option. When used in this circumstance, the RESTART DATABASE command will resume I/O writes without performing crash recovery. The RESTART DATABASE command with the WRITE RESUME option will only perform crash recovery when you use it after a database crash.

START DATABASE MANAGER

Starts the current database manager instance background processes on a single database partition or on all the database partitions defined in a multi-partitioned database environment.

Scope

In a multi-partitioned database environment, this command affects all database partitions that are listed in the `$HOME/sqllib/db2nodes.cfg` file, unless the DBPARTITIONNUM parameter is used.

Authorization

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

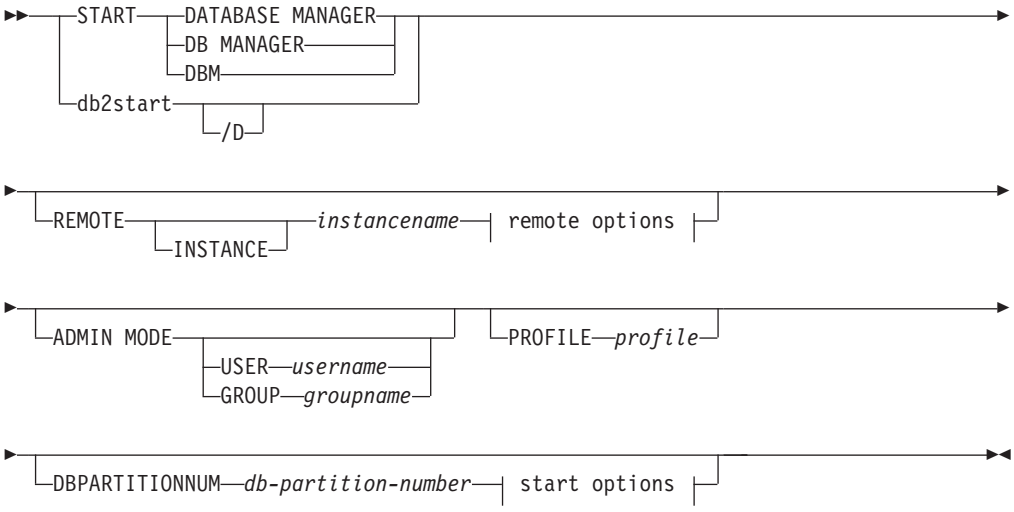
The ADD DBPARTITIONNUM start option requires either *sysadm* or *sysctrl* authority.

You must meet Windows operating system requirements for starting a service. If Extended Security is disabled, you must be a member of the Administrators, Server Operators or Power Users group. If Extended Security is enabled, you must be a member of either the Administrators group or the DB2ADMNS group to start the database.

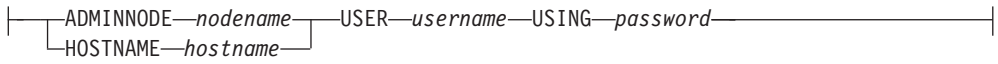
Required connection

None

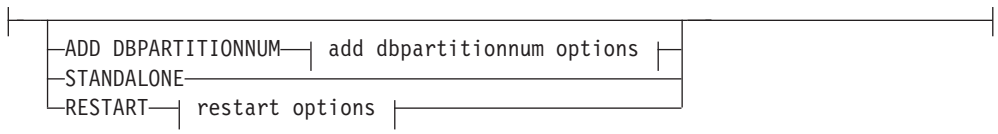
Command syntax



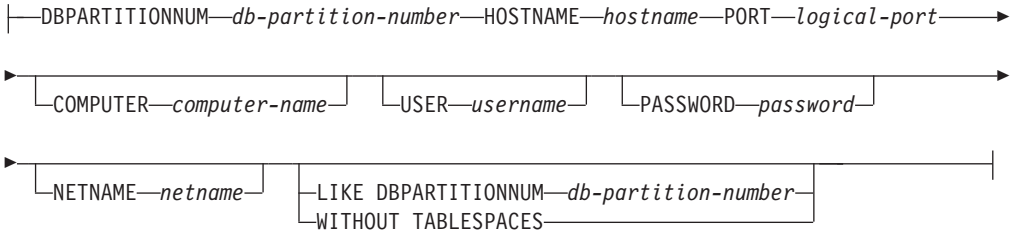
remote options:



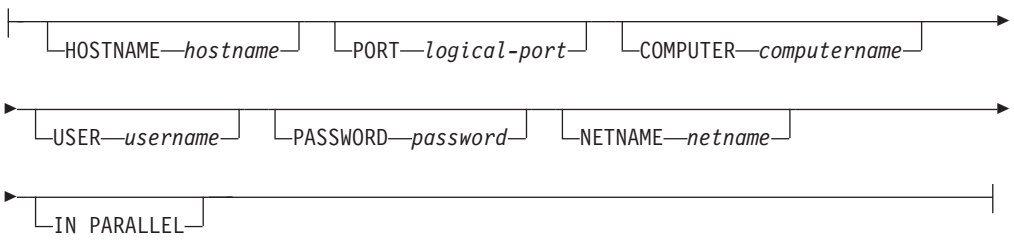
start options:



add dbpartitionnum options:



restart options:



Command parameters

/D Allows the DB2 product installation on Windows to be run as a process.

REMOTE [INSTANCE] *instancename*
Specifies the name of the remote instance you want to start.

ADMINNODE *nodename*
With REMOTE, or REMOTE INSTANCE, specifies the name of the administration node.

HOSTNAME *hostname*
With REMOTE, or REMOTE INSTANCE, specifies the name of the host node.

USER *username*
With REMOTE, or REMOTE INSTANCE, specifies the name of the user.

USING *password*
With REMOTE, or REMOTE INSTANCE, and USER, specifies the password of the user.

ADMIN MODE
Starts the instance in quiesced mode for administration purposes. This is equivalent to the QUIESCE INSTANCE command except in this case the instance is not already "up", and therefore there is no need to force the connections OFF.

USER *username*
With ADMIN MODE, specifies the name of the user.

GROUP *groupname*
With ADMIN MODE, specifies the name of the group.

All of the following parameters are valid in an Enterprise Server Edition (ESE) environment only.

PROFILE *profile*
Specifies the name of the profile file to be executed at each database partition to define the DB2 environment. This file is executed before the database partitions are started. The profile file must reside in the `sqllib` directory of the instance owner. The environment variables in the profile file are not necessarily all defined in the user session.

DBPARTITIONNUM *db-partition-number*
Specifies the database partition to be started. If no other options are specified, a normal startup is done at this database partition.

Valid values are from 0 to 999 inclusive. If ADD DBPARTITIONNUM is not specified, the value must already exist in the `db2nodes.cfg` file of the instance owner. If no database partition number is specified, all database partitions defined in the configuration file are started.

ADD DBPARTITIONNUM
Specifies that the new database partition server is added to the `db2nodes.cfg` file of the instance owner with the *hostname* and *logical-port* values.

Ensure that the combination of *hostname* and *logical-port* is unique.

The add database partition server utility is executed internally to create all existing databases on the database server partition being added. The new database partition server is automatically added to the db2nodes.cfg file.

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

If the ADD request is made in an environment that has two or more active database partition servers, the new database partition server is visible to the environment when the ADD processing completes.

If the ADD request is made in an environment that has one database partition server and it is active, after ADD processing completes, the new database partition server is inactive. The instance must be restarted by using db2stop and db2start before the new database partition server can participate in the partitioned database environment. If the ADD request is made in an environment that has one database partition server and it is inactive, after ADD processing completes, the new database partition server (or servers, if more than one is added) is active. Only the original database partition server needs to be started.

A newly added database partition is configured during ADD processing as follows:

1. In a multipartition environment, the new database partition is configured using the database configuration parameter values from a noncatalog database partition.
2. In a single-partition environment, the new database partition is configured using the database configuration parameter values from the catalog partition.
3. If a problem occurs in copying the database configuration parameter values to the new database partition, the new database partition is configured using the default database configuration parameter values.

DBPARTITIONNUM *db-partition-number*
Specifies the value of the database partition number for the new database partition to be created.

HOSTNAME *hostname*
With ADD DBPARTITIONNUM, specifies the host name to be added to the db2nodes.cfg file.

PORT *logical-port*
With ADD DBPARTITIONNUM, specifies the logical port to be added to the db2nodes.cfg file. Valid values are from 0 to 999.

COMPUTER computername
The computer name for the machine on which the new database partition is created. This parameter is mandatory on Windows, but is ignored on other operating systems.

USER *username*
The user name for the account on the new database partition. This parameter is mandatory on Windows, but is ignored on other operating systems.

PASSWORD *password*
The password for the account on the new database partition. This parameter is mandatory on Windows, but is ignored on other operating systems.

NETNAME *netname*
Specifies the *netname* to be added to the `db2nodes.cfg` file. If not specified, this parameter defaults to the value specified for *hostname*.

IN PARALLEL
Issue the RESTART DATABASE command for parallel execution.

LIKE DBPARTITIONNUM *db-partition-number*
Specifies that the containers for the system temporary table spaces will be the same as the containers on the specified *db-partition-number* for each database in the instance. The database partition specified must be a database partition that is already in the `db2nodes.cfg` file. For system temporary table spaces that are defined to use automatic storage (in other words, system temporary table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement or where no MANAGED BY CLAUSE was specified at all), the containers will not necessarily match those from the partition specified. Instead, containers will automatically be assigned by the database manager based on the storage paths that are associated with the database. This may or may not result in the same containers being used on these two partitions.

WITHOUT TABLESPACES
Specifies that containers for the system temporary table spaces are not created for any of the databases. The ALTER TABLESPACE statement must be used to add system temporary table space containers to each database before the database can be used. This option is ignored for system temporary table spaces that are defined to use automatic storage (in other words, system temporary table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement or where no MANAGED BY CLAUSE was specified at all). For these table spaces, there is no way to defer container creation. Containers will automatically be assigned by the database manager based on the storage paths that are associated with the database.

STANDALONE
Specifies that the database partition is to be started in stand-alone mode. FCM does not attempt to establish a connection to any other database partition. This option is used when adding a database partition.

RESTART
Starts the database manager after a failure. Other database partitions are still operating, and this database partition attempts to connect to the others. If neither the *hostname* nor the *logical-port* parameter is specified, the database manager is restarted using the *hostname* and *logical-port* values specified in `db2nodes.cfg`. If either parameter is specified, the new values are sent to the other database partitions when a connection is established. The `db2nodes.cfg` file is updated with this information.

HOSTNAME *hostname*
You can use the HOSTNAME option with the RESTART parameter to restart a database partition on a different machine than is specified in the database partition configuration file, `db2nodes.cfg`.

Restriction:
When you are using the DB2 High Availability Feature, you should not use the HOSTNAME option with the RESTART parameter to restart a database partition on a different machine. To restart or move a database partition from one machine in a cluster to another machine, use DB2 High Availability Instance Configuration Utility (db2haicu).

PORT *logical-port*
With RESTART, specifies the logical port number to be used to override that in the database partition configuration file. If not specified, this parameter defaults to the *logical-port* value that corresponds to the *num* value in the `db2nodes.cfg` file. Valid values are from 0 to 999.

COMPUTER *computername*
The computer name for the machine on which the new database partition is created. This parameter is mandatory on Windows, but is ignored on other operating systems.

USER *username*
The user name for the account on the new database partition. This parameter is mandatory on Windows, but is ignored on other operating systems.

PASSWORD *password*
The password for the account on the new database partition. This parameter is mandatory on Windows, but is ignored on other operating systems.

NETNAME *netname*
Specifies the *netname* to override that specified in the `db2nodes.cfg` file. If not specified, this parameter defaults to the *netname* value that corresponds to the *db-partition-number* value in the `db2nodes.cfg` file.

Examples

The following is sample output from db2start issued on a three-database partition system with database partitions 10, 20, and 30:

```
04-07-1997 10:33:05    10   0   SQL1063N  DB2START processing was successful.
04-07-1997 10:33:07    20   0   SQL1063N  DB2START processing was successful.
04-07-1997 10:33:07    30   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.
```

Usage notes

On Microsoft Windows Vista or later versions, you must execute this command from a DB2 command window running with full administrator privileges.

It is not necessary to issue this command on a client node. It is provided for compatibility with older clients, but it has no effect on the database manager.

Once started, the database manager instance runs until the user stops it, even if all application programs that were using it have ended.

If the database manager starts successfully, a successful completion message is sent to the standard output device. If an error occurs, processing stops, and an error message is sent to the standard output device. In a partitioned database environment, messages are returned on the database partition that issued the START DATABASE MANAGER command.

If no parameters are specified in a partitioned database environment, the database manager is started on all parallel nodes using the parameters specified in the database partition configuration file.

If a START DATABASE MANAGER command is in progress, ensure that the applicable database partitions have started *before* issuing a request to the database.

The `db2cshrc` file is not supported and cannot be used to define the environment.

You can start an instance in a quiesced state. You can do this by using one of the following choices:

```
db2start admin mode
```

or

```
db2start admin mode user username
```

or

```
db2start admin mode group groupname
```

When adding a new database partition server, START DATABASE MANAGER 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 then 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, START DATABASE MANAGER might have to communicate with another database partition server to retrieve the table space definitions for the database partitions that reside on that server. The **start_stop_time** database manager configuration parameter is used to specify the time, in minutes, by which the other database partition server must respond with the automatic storage and table space definitions. If this time is exceeded, the command fails. If this situation occurs, increase the value of **start_stop_time**, and reissue the command.

A new database partition server cannot be added when any of the following commands, statements, or operations are in progress. Otherwise SQL6074N is returned.

- QUIESCE INSTANCE
- UNQUIESCE INSTANCE
- STOP DB2 (db2stop)
- STOP DATABASER MANAGER DBPARTITIONNUM
- START DB2 (db2start)
- START DATABASE MANAGER DBPARTITIONNUM
- START DATABASE MANAGER with restart options
- CREATE DATABASE

- DROP DATABASE
- QUIESCE DATABASE
- UNQUIESCE DATABASE
- ACTIVATE DATABASE
- DEACTIVATE DATABASE
- A Z lock on a database object
- Backing up the database on all database partition servers
- Restoring the database
- ALTER, ALTER, or DROP of a table space
- Updating of automatic storage paths

On UNIX platforms, the START DATABASE MANAGER command supports the SIGINT signal. It is issued if CTRL+C is pressed. If this signal occurs, all in-progress startups are interrupted and a message (SQL1044N) is returned from each interrupted database partition to the `$HOME/sqllib/log/db2start.`*timestamp*`.log` error log file. Database partitions that are already started are not affected. If CTRL+C is pressed on a database partition that is starting, db2stop must be issued on that database partition before an attempt is made to start it again.

On Windows operating systems, neither the db2start command nor the NET START command returns warnings if any communication subsystem failed to start. The database manager in a Windows environment is implemented as a service, and does not return an error if the service is started successfully. Be sure to examine the Event Log or the db2diag log file for any errors that might have occurred during the running of db2start.

Compatibilities

For compatibility with versions earlier than Version 8:

- The keywords LIKE NODE can be substituted for LIKE DBPARTITIONNUM.
- The keyword ADDNODE can be substituted for ADD DBPARTITIONNUM.
- The keyword NODENUM can be substituted for DBPARTITIONNUM.

STOP DATABASE MANAGER

Stops the current database manager instance. Unless explicitly stopped, the database manager continues to be active. This command does not stop the database manager instance if any applications are connected to databases. If there are no database connections, but there are instance attachments, it forces the instance attachments and stops the database manager. This command also deactivates any outstanding database activations before stopping the database manager.

In a partitioned database environment, this command stops the current database manager instance on a database partition or on all database partitions. When it stops the database manager on all database partitions, it uses the `db2nodes.cfg` configuration file to obtain information about each database partition.

This command can also be used to drop a database partition from the `db2nodes.cfg` file (partitioned database environments only).

This command is not valid on a client.

Scope

By default, and in a partitioned database environment, this command affects all database partitions that are listed in the `db2nodes.cfg` file.

Authorization

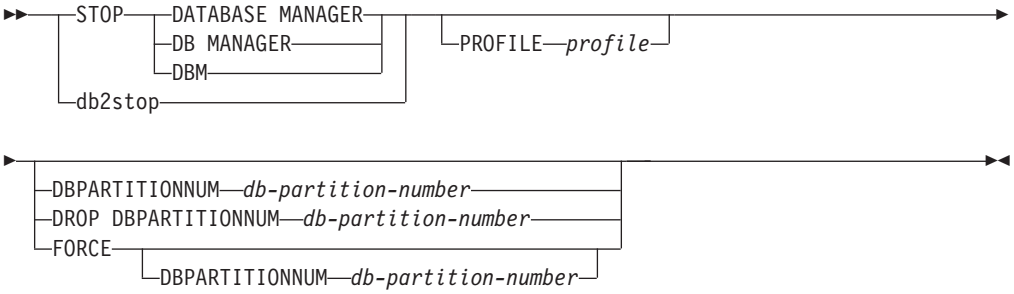
One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection

None

Command syntax



Command parameters

PROFILE *profile*
Partitioned database environments only. Specifies the name of the profile file that was executed at startup to define the DB2 environment for those database partitions that were started. If a profile for the START DATABASE MANAGER command was specified, the same profile must be specified here. The profile file must reside in the `sqllib` directory of the instance owner.

DBPARTITIONNUM *db-partition-number*
Partitioned database environments only. Specifies the database partition to be stopped.

Valid values are from 0 to 999 inclusive, and must be in the `db2nodes.cfg` file. If no database partition number is specified, all database partitions defined in the configuration file are stopped.

DROP DBPARTITIONNUM *db-partition-number*
Partitioned database environments only. Specifies the database partition to be dropped from the `db2nodes.cfg` file.

Before using this parameter, run the DROP DBPARTITIONNUM VERIFY command to ensure that there is no user data on this database partition.

When this option is specified, all database partitions in the `db2nodes.cfg` file are stopped.

FORCE
Specifies to use FORCE APPLICATION ALL when stopping the database manager at each database partition.

DBPARTITIONNUM *db-partition-number*
Partitioned database environments only. Specifies the database partition to be stopped after all applications on that database partition have been forced to stop. If the FORCE option is used without this parameter, all applications on all database partitions are forced before all the database partitions are stopped.

Examples

The following is sample output from db2stop issued on a three-partition system with database partitions 10, 20, and 30:

```
04-07-1997 10:32:53    10   0   SQL1064N  DB2STOP processing was successful.
04-07-1997 10:32:54    20   0   SQL1064N  DB2STOP processing was successful.
04-07-1997 10:32:55    30   0   SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.
```

Usage notes

On Microsoft Windows Vista or later versions, you must execute this command from a DB2 command window running with full administrator privileges.

It is not necessary to issue this command on a client node. It is provided for compatibility with older clients, but it has no effect on the database manager.

Once started, the database manager instance runs until the user stops it, even if all application programs that were using it have ended.

If the database manager is stopped, a successful completion message is sent to the standard output device. If an error occurs, processing stops, and an error message is sent to the standard output device.

If the database manager cannot be stopped because application programs are still connected to databases, use the FORCE APPLICATION command to disconnect all users first, or reissue the STOP DATABASE MANAGER command with the FORCE option.

The following information applies to partitioned database environments only:

- If no parameters are specified, the database manager is stopped on each database partition listed in the configuration file. The administration notification log might contain messages to indicate that other database partitions are shutting down.
- Any database partitions added to the partitioned database environment since the previous STOP DATABASE MANAGER command was issued will be updated in the `db2nodes.cfg` file.
- On UNIX platforms, if the value specified for the **start_stop_time**database manager configuration parameter is reached, all in-progress stops are interrupted, and message SQL6037N is returned from each interrupted database partition to the `$HOME/sqllib/log/db2stop.` *timestamp*`.log` error log file. Database partitions that are already stopped are not affected.
- The `db2cshrc` file is not supported and cannot be specified as the value for the PROFILE parameter.

Attention: The UNIX kill command should *not* be used to terminate the database manager because it will abruptly end database manager processes without controlled termination and cleanup processing.

UNQUIESCE

Unless specifically designated, no user except those with *sysadm*, *sysmaint*, or *sysctrl* has access to a database while it is quiesced. Therefore an UNQUIESCE is required to restore general access to a quiesced database.

Scope

UNQUIESCE DB restores user access to all objects in the quiesced database.

To stop the instance and unquiesce it and all its databases, issue the db2stop command. Stopping and restarting DB2 will unquiesce all instances and databases.

Authorization

One of the following:

For database level unquiesce:

- *sysadm*
- *dbadm*

Command syntax

Required connection

Database

Command parameters

DB Unquiesce the database. User access will be restored to all objects in the database.

Examples

Unquiescing a Database

This command will unquiesce the database that had previously been quiesced.

UNQUIESCE

Unless specifically designated, no user except those with *sysadm*, *sysmaint*, or *sysctrl* has access to a database while it is quiesced. Therefore an UNQUIESCE is required to restore general access to a quiesced database.

Scope

UNQUIESCE DB restores user access to all objects in the quiesced database.

To stop the instance and unquiesce it and all its databases, issue the db2stop command. Stopping and restarting DB2 will unquiesce all instances and databases.

Authorization

One of the following:

For database level unquiesce:

- *sysadm*
- *dbadm*

Command syntax

Required connection

Database

Command parameters

DB Unquiesce the database. User access will be restored to all objects in the database.

Examples

Unquiescing a Database

This command will unquiesce the database that had previously been quiesced.

UPDATE DATABASE CONFIGURATION

Modifies individual entries in a specific database configuration file.

A database configuration file resides on every database partition on which the database has been created.

Scope

This command updates all database partitions by default, except when DBPARTITIONNUM is specified to update only one database partition.

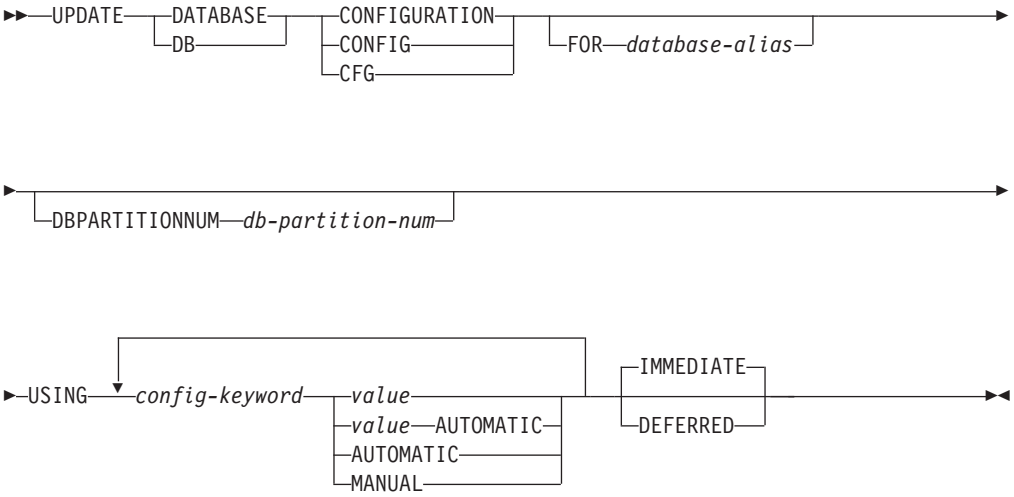
Authorization

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection

Command syntax



Command parameters

AUTOMATIC
Some configuration parameters can be set to AUTOMATIC, allowing DB2 to automatically adjust these parameters to reflect the current resource requirements. For a list of configuration parameters that support the AUTOMATIC keyword, refer to the configuration parameters summary. If a value is specified along with the AUTOMATIC keyword, it might influence the automatic calculations. For specific details about this behavior, refer to the documentation for the configuration parameter.

DEFERRED
Make the changes only in the configuration file, so that the changes take effect the next time you reactivate the database.

FOR *database-alias*
Specifies the alias of the database whose configuration is to be updated. Specifying the database alias is not required when a database connection has already been established. You can update the configuration file for another database residing under the same database instance. For example, if you are connected only to database `db11`, and issue `update db config for alias db22 using .... immediate`:

- If there is no active connection on `db22`, the update will be successful because only the configuration file needs to be updated. A new connection (which will activate the database) will see the new change in memory.
- If there are active connections on `db22` from other applications, the update will work on disk but not in memory. You will receive a warning saying that the database needs to be restarted.

DBPARTITIONNUM *db-partition-num*
If a database configuration update is to be applied to a specific database partition, this parameter may be used. If this parameter is not provided, the update will take effect on all database partitions.

IMMEDIATE
Make the changes immediately, while the database is running.

This is a default clause when operating in the CLPPlus interface as well. IMMEDIATE need not be called when using CLPPlus processor.

MANUAL
Disables automatic tuning for the configuration parameter. The parameter is set to its current internal value and is no longer updated automatically.

USING *config-keyword value*
config-keyword specifies the database configuration parameter to be updated. *value* specifies the value to be assigned to the parameter.

Usage notes

For more information about DB2 configuration parameters and the values available for each type of database node, see the individual configuration parameter descriptions. The values of these parameters differ for each type of database node configured (server, client, or server with remote clients).

Not all parameters can be updated.

Some changes to the database configuration file become effective only after they are loaded into memory. All applications must disconnect from the database before this can occur. For more information on which parameters are configurable on-line and which ones are not, see summary list of configuration parameters.

If an error occurs, the database configuration file does not change. The database configuration file cannot be updated if the checksum is invalid. This might occur if the database configuration file is changed without using the appropriate command. If this happens, the database must be restored to reset the database configuration file.

UPDATE DATABASE CONFIGURATION

Modifies individual entries in a specific database configuration file.

A database configuration file resides on every database partition on which the database has been created.

Scope

This command updates all database partitions by default, except when DBPARTITIONNUM is specified to update only one database partition.

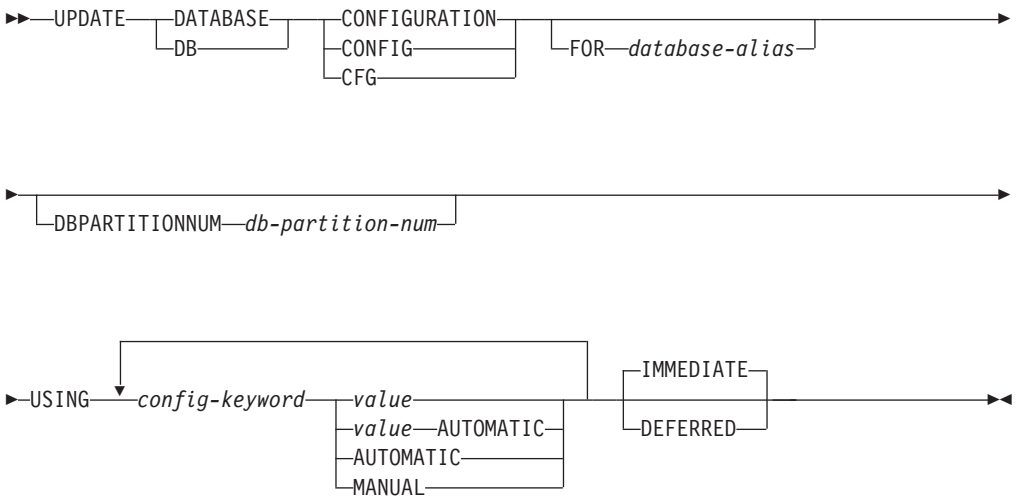
Authorization

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection

Command syntax



Command parameters

AUTOMATIC
Some configuration parameters can be set to AUTOMATIC, allowing DB2 to automatically adjust these parameters to reflect the current resource requirements. For a list of configuration parameters that support the AUTOMATIC keyword, refer to the configuration parameters summary. If a value is specified along with the AUTOMATIC keyword, it might influence the automatic calculations. For specific details about this behavior, refer to the documentation for the configuration parameter.

DEFERRED
Make the changes only in the configuration file, so that the changes take effect the next time you reactivate the database.

FOR *database-alias*
Specifies the alias of the database whose configuration is to be updated. Specifying the database alias is not required when a database connection has already been established. You can update the configuration file for another database residing under the same database instance. For example, if you are connected only to database `db11`, and issue `update db config for alias db22 using .... immediate`:

- If there is no active connection on `db22`, the update will be successful because only the configuration file needs to be updated. A new connection (which will activate the database) will see the new change in memory.
- If there are active connections on `db22` from other applications, the update will work on disk but not in memory. You will receive a warning saying that the database needs to be restarted.

DBPARTITIONNUM *db-partition-num*
If a database configuration update is to be applied to a specific database partition, this parameter may be used. If this parameter is not provided, the update will take effect on all database partitions.

IMMEDIATE
Make the changes immediately, while the database is running.

This is a default clause when operating in the CLPPlus interface as well. IMMEDIATE need not be called when using CLPPlus processor.

MANUAL
Disables automatic tuning for the configuration parameter. The parameter is set to its current internal value and is no longer updated automatically.

USING *config-keyword value*
config-keyword specifies the database configuration parameter to be updated. *value* specifies the value to be assigned to the parameter.

Usage notes

For more information about DB2 configuration parameters and the values available for each type of database node, see the individual configuration parameter descriptions. The values of these parameters differ for each type of database node configured (server, client, or server with remote clients).

Not all parameters can be updated.

Some changes to the database configuration file become effective only after they are loaded into memory. All applications must disconnect from the database before this can occur. For more information on which parameters are configurable on-line and which ones are not, see summary list of configuration parameters.

If an error occurs, the database configuration file does not change. The database configuration file cannot be updated if the checksum is invalid. This might occur if the database configuration file is changed without using the appropriate command. If this happens, the database must be restored to reset the database configuration file.

UPDATE DATABASE MANAGER CONFIGURATION

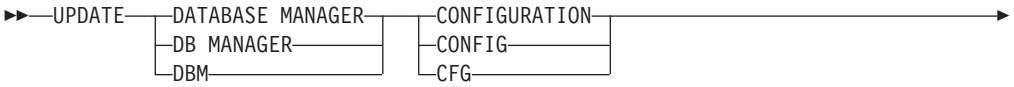
Modifies individual entries in the database manager configuration file.

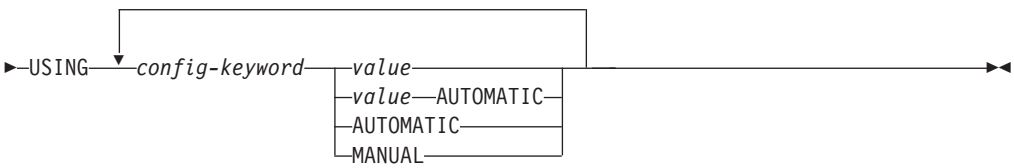
Authorization

sysadm

Required connection

Command syntax





Command parameters

AUTOMATIC
Some configuration parameters can be set to AUTOMATIC, allowing DB2 to automatically adjust these parameters to reflect the current resource requirements. For a list of configuration parameters that support the AUTOMATIC keyword, refer to the configuration parameters summary. If a value is specified along with the AUTOMATIC keyword, it might influence the automatic calculations. For specific details about this behavior, refer to the documentation for the configuration parameter.

Note:

DEFERRED
Make the changes only in the configuration file, so that the changes take effect when the instance is restarted.

This is a default clause when operating in the CLPPlus interface. DEFERRED need not be called when using CLPPlus processor.

MANUAL
Disables automatic tuning for the configuration parameter. The parameter is set to its current internal value and is no longer updated automatically.

USING *config-keyword value*
Specifies the database manager configuration parameter to be updated. For a list of configuration parameters, refer to the configuration parameters summary. *value* specifies the value to be assigned to the parameter.

Usage notes

Not all parameters can be updated.

Some changes to the database manager configuration file become effective only after they are loaded into memory. For more information on which parameters are configurable online and which ones are not, see the configuration parameter summary. Server configuration parameters that are not reset immediately are reset during execution of db2start. For a client configuration parameter, parameters are reset the next time you restart the application. If the client is the command line processor, it is necessary to invoke TERMINATE.

If an error occurs, the database manager configuration file does not change.

The database manager configuration file cannot be updated if the checksum is invalid. This can occur if you edit database manager configuration file and do not use the appropriate command. If the checksum is invalid, you must reinstall the database manager to reset the database manager configuration file.

When you update the **SVCENAME**, or **TPNAME** database manager configuration parameters for the current instance, if LDAP support is enabled and there is an LDAP server registered for this instance, the LDAP server is updated with the new value or values.

UPDATE DATABASE MANAGER CONFIGURATION

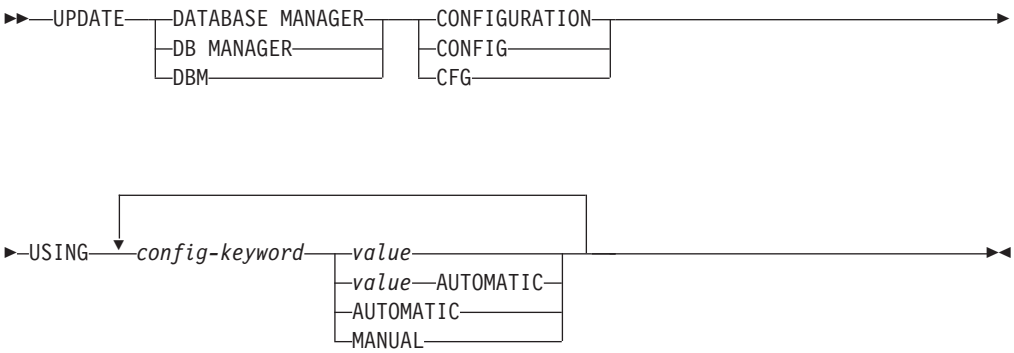
Modifies individual entries in the database manager configuration file.

Authorization

sysadm

Required connection

Command syntax



Command parameters

AUTOMATIC
Some configuration parameters can be set to AUTOMATIC, allowing DB2 to automatically adjust these parameters to reflect the current resource requirements. For a list of configuration parameters that support the AUTOMATIC keyword, refer to the configuration parameters summary. If a value is specified along with the AUTOMATIC keyword, it might influence the automatic calculations. For specific details about this behavior, refer to the documentation for the configuration parameter.

Note:

DEFERRED
Make the changes only in the configuration file, so that the changes take effect when the instance is restarted.

This is a default clause when operating in the CLPPlus interface. DEFERRED need not be called when using CLPPlus processor.

MANUAL
Disables automatic tuning for the configuration parameter. The parameter is set to its current internal value and is no longer updated automatically.

USING *config-keyword value*
Specifies the database manager configuration parameter to be updated. For a list of configuration parameters, refer to the configuration parameters summary. *value* specifies the value to be assigned to the parameter.

Usage notes

Not all parameters can be updated.

Some changes to the database manager configuration file become effective only after they are loaded into memory. For more information on which parameters are configurable online and which ones are not, see the configuration parameter summary. Server configuration parameters that are not reset immediately are reset

during execution of db2start. For a client configuration parameter, parameters are reset the next time you restart the application. If the client is the command line processor, it is necessary to invoke TERMINATE.

If an error occurs, the database manager configuration file does not change.

The database manager configuration file cannot be updated if the checksum is invalid. This can occur if you edit database manager configuration file and do not use the appropriate command. If the checksum is invalid, you must reinstall the database manager to reset the database manager configuration file.

When you update the **SVCENAME**, or **TPNAME** database manager configuration parameters for the current instance, if LDAP support is enabled and there is an LDAP server registered for this instance, the LDAP server is updated with the new value or values.

Commands for Users

ATTACH

Enables an application to specify the instance at which instance-level commands (CREATE DATABASE and FORCE APPLICATION, for example) are to be executed. This instance can be the current instance, another instance on the same workstation, or an instance on a remote workstation.

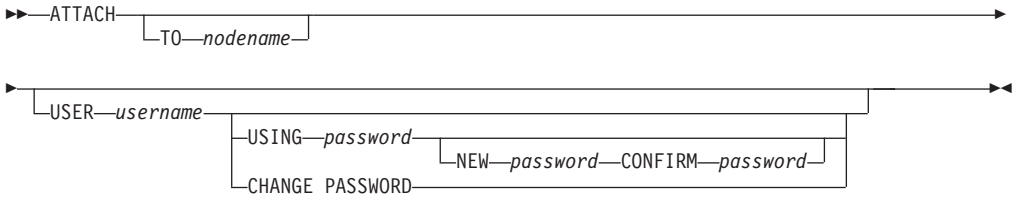
Authorization

None

Required connection

None. This command establishes an instance attachment.

Command syntax



Command parameters

TO *nodename*
Alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception to this is the local instance (as specified by the **DB2INSTANCE** environment variable) which can be specified as the object of an attach, but which cannot be used as a node name in the node directory.

USER *username*
Specifies the authentication identifier. When attaching to a DB2 database instance on a Windows operating system, the user name can be specified in a format compatible with Microsoft Security Account Manager (SAM).

The qualifier must be a flat-style (NetBIOS-like) name, which has a maximum length of 15 characters. For example, *domainname\username*.

USING *password*
Specifies the password for the user name. If a user name is specified, but a password is *not* specified, the user is prompted for the current password. The password is not displayed at entry.

NEW *password*
Specifies the new password that is to be assigned to the user name. The system on which the password will be changed depends on how user authentication has been set up. The DB2 database system provides support for changing passwords on AIX, Linux and Windows operating systems, and supports up to 255 characters for your own written plugins. See *Password rules* for additional information about passwords.

CONFIRM *password*
A string that must be identical to the new password. This parameter is used to catch entry errors.

CHANGE PASSWORD
If this option is specified, the user is prompted for the current password, a new password, and for confirmation of the new password. Passwords are not displayed at entry.

Examples

Catalog two remote nodes:

```
db2 catalog tcpip node node1 remote freedom server server1
db2 catalog tcpip node node2 remote flash server server1
```

Attach to the first node, force all users, and then detach:

```
db2 attach to node1
db2 force application all
db2 detach
```

Attach to the second node, and see who is on:

```
db2 attach to node2
db2 list applications
```

After the command returns agent IDs 1, 2 and 3, force 1 and 3, and then detach:

```
db2 force application (1, 3)
db2 detach
```

Attach to the current instance (not necessary, will be implicit), force all users, then detach (AIX only):

```
db2 attach to $DB2INSTANCE
db2 force application all
db2 detach
```

Usage notes

If *nodename* is omitted from the command, information about the current state of attachment is returned.

If ATTACH has not been executed, instance-level commands are executed against the current instance, specified by the **DB2INSTANCE** environment variable.

DETACH

Removes the logical DBMS instance attachment, and terminates the physical communication connection if there are no other logical connections using this layer.

Authorization

None

Required connection

None. Removes an existing instance attachment.

Command syntax

Command parameters

None

GET CONNECTION STATE

Displays the connection state. Possible states are:

- Connectable and connected
- Connectable and unconnected
- Unconnectable and connected
- Implicitly connectable (if implicit connect is available).

This command also returns information about:

- the database connection mode (SHARE or EXCLUSIVE)
- the alias and name of the database to which a connection exists (if one exists)
- the host name and service name of the connection if the connection is using TCP/IP

Authorization

None

Required connection

None

Command syntax

Command parameters

None

Examples

The following is sample output from GET CONNECTION STATE:

```
   Database Connection State

 Connection state       = Connectable and Connected
 Connection mode        = SHARE
 Local database alias   = SAMPLE
 Database name          = SAMPLE
 Hostname               = montero
 Service name           = 29384
```

Usage notes

This command does not apply to type 2 connections.

LIST DBPARTITIONNUMS

Lists all database partitions associated with the current database.

Scope

This command can be issued from any database partition that is listed in `$HOME/sqllib/db2nodes.cfg`. It returns the same information from any of these database partitions.

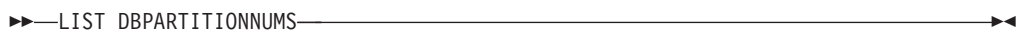
Authorization

None

Required connection

Database

Command syntax



Command parameters

None

Examples

Following is sample output from the LIST DBPARTITIONNUMS command:

```
DATABASE PARTITION NUMBER
-------------------------
                        0
                        2
                        5
                        7
                        9

     5 record(s) selected.
```

Compatibilities

For compatibility with versions earlier than Version 8:

- The keyword NODES can be substituted for DBPARTITIONNUMS.

PRECOMPILE

Processes an application program source file containing embedded SQL statements. A modified source file is produced, containing host language calls for the SQL statements and, by default, a package is created in the database.

Scope

This command can be issued from any database partition in `db2nodes.cfg`. In a partitioned database environment, it can be issued from any database partition server defined in the `db2nodes.cfg` file. It updates the database catalogs on the catalog database partition. Its effects are visible to all database partitions.

Authorization

One of the following authorizations:

- *dbadm* authority
- If EXPLAIN ONLY is specified, EXPLAIN authority or an authority that implicitly includes EXPLAIN is sufficient.
- If a package does not exist, BINDADD authority and:
  - If the schema name of the package does not exist, IMPLICIT_SCHEMA authority on the database.
  - If the schema name of the package does exist, CREATEIN privilege on the schema.
- If the package exists, one of the following privileges:
  - ALTERIN privilege on the schema
  - BIND privilege on the package

In addition, if capturing explain information using the EXPLAIN or the EXPLSNAP clause, one of the following authorizations is required:

- INSERT privilege on the explain tables
- DATAACCESS authority

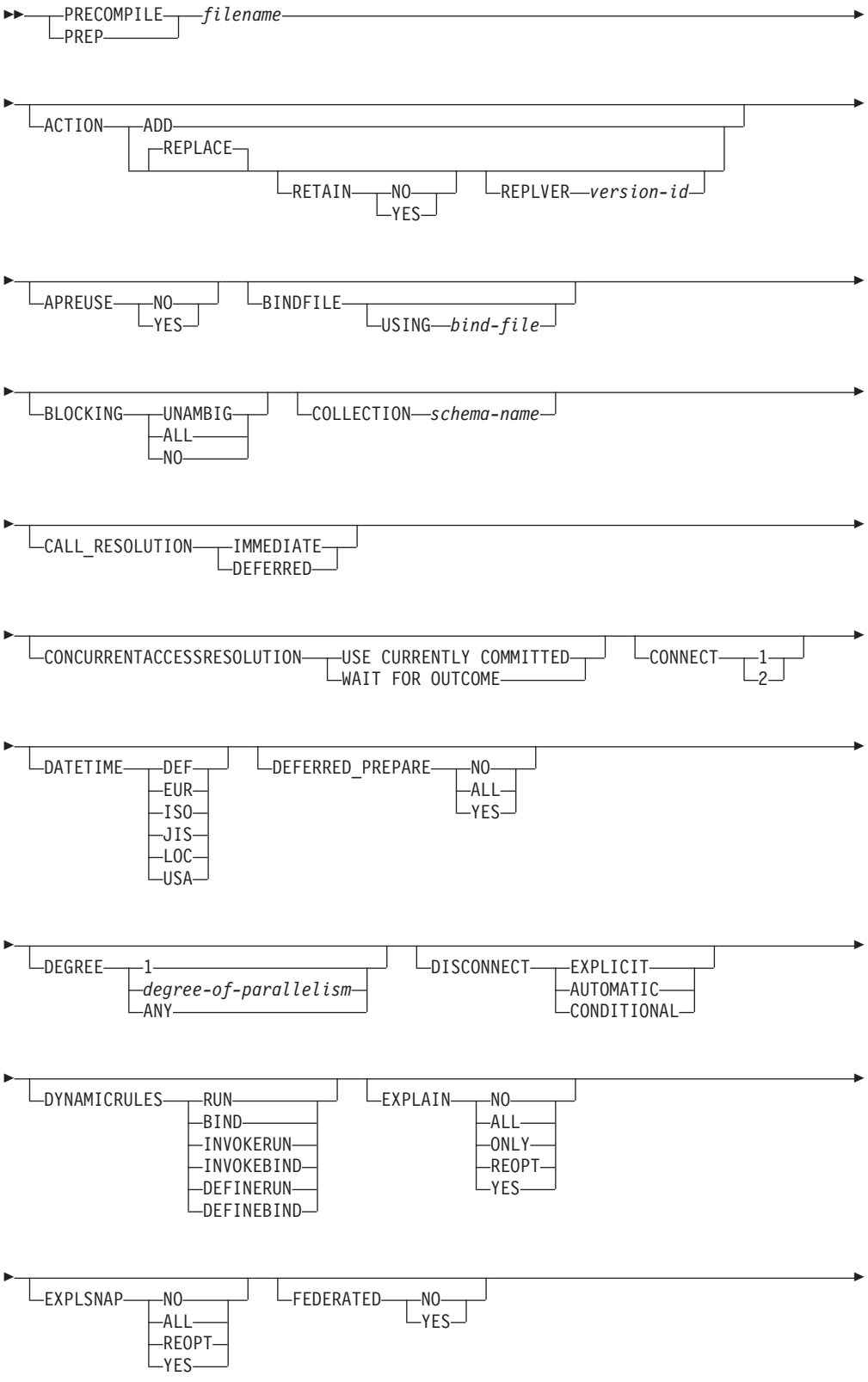
The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements.

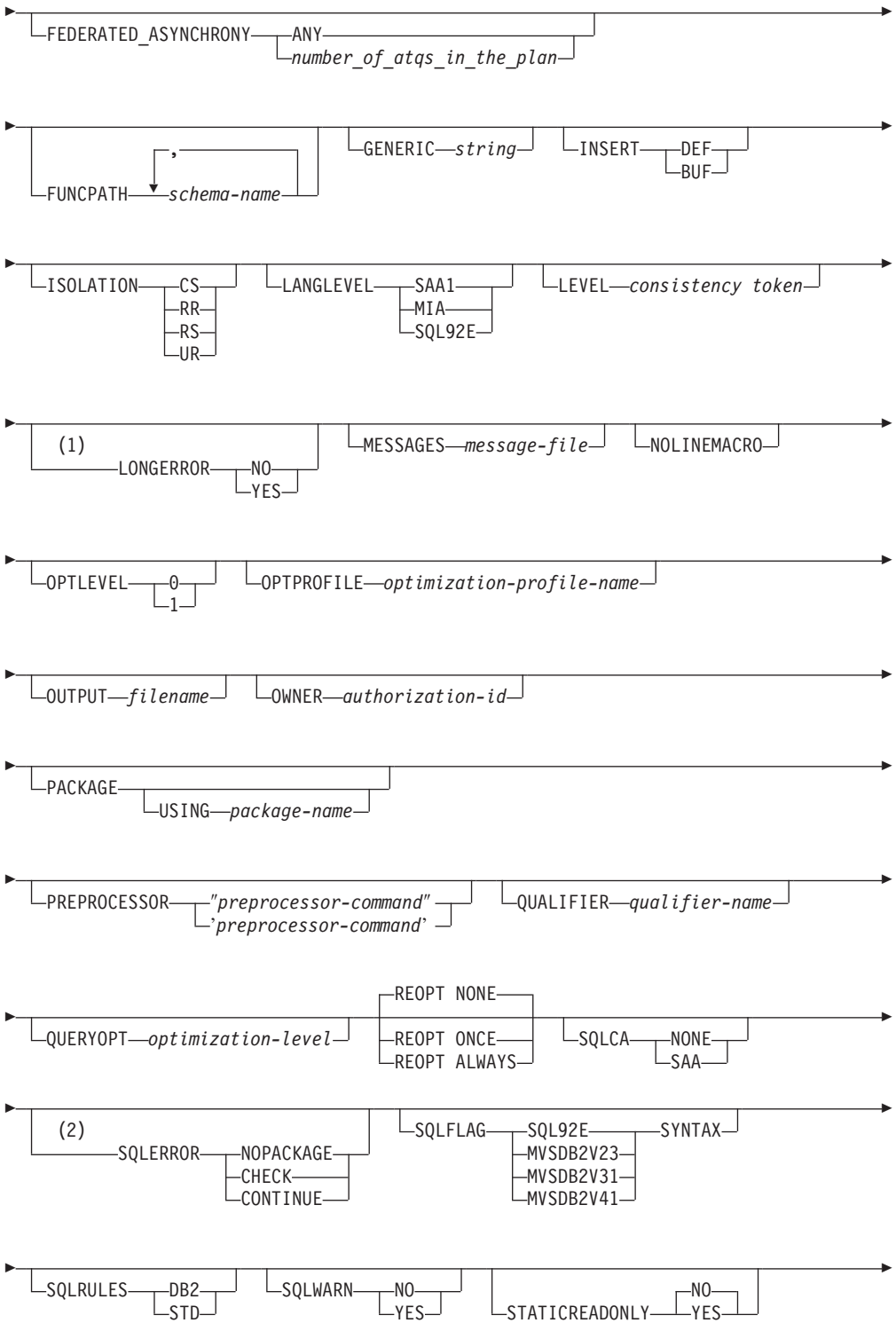
Required connection

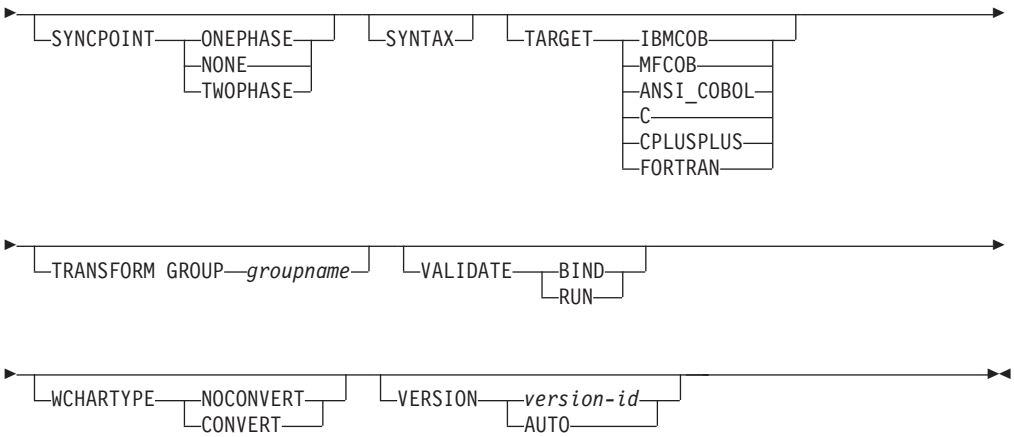
Database. If implicit connect is enabled, a connection to the default database is established.

Command syntax

For DB2 Database for Linux, UNIX, and Windows


PRECOMPILE
PREP
filename
ACTION
ADD
REPLACE
RETAIN
NO
YES
REPLVER
version-id
APREUSE
NO
YES
BINDFILE
USING
bind-file
BLOCKING
UNAMBIG
ALL
NO
COLLECTION
schema-name
CALL_RESOLUTION
IMMEDIATE
DEFERRED
CONCURRENTACCESSRESOLUTION
USE CURRENTLY COMMITTED
WAIT FOR OUTCOME
CONNECT
1
2
DATETIME
DEF
EUR
ISO
JIS
LOC
USA
DEFERRED_PREPARE
NO
ALL
YES
DEGREE
1
degree-of-parallelism
ANY
DISCONNECT
EXPLICIT
AUTOMATIC
CONDITIONAL
DYNAMICRULES
RUN
BIND
INVOKERUN
INVOKEBIND
DEFINERUN
DEFINEBIND
EXPLAIN
NO
ALL
ONLY
REOPT
YES
EXPLSNAP
NO
ALL
REOPT
YES
FEDERATED
NO
YES


FEDERATED_ASYNCHRONY
ANY
number_of_atqs_in_the_plan
FUNCPATH
,
schema-name
GENERIC
string
INSERT
DEF
BUF
ISOLATION
CS
RR
RS
UR
LANGLEVEL
SAA1
MIA
SQL92E
LEVEL
consistency token
(1)
LONGERROR
NO
YES
MESSAGES
message-file
NOLINEMACRO
OPTLEVEL
0
1
OPTPROFILE
optimization-profile-name
OUTPUT
filename
OWNER
authorization-id
PACKAGE
USING
package-name
PREPROCESSOR
"preprocessor-command"
'preprocessor-command'
QUALIFIER
qualifier-name
QUERYOPT
optimization-level
REOPT NONE
REOPT ONCE
REOPT ALWAYS
SQLCA
NONE
SAA
(2)
SQLERROR
NOPACKAGE
CHECK
CONTINUE
SQLFLAG
SQL92E
SYNTAX
MVSDB2V23
MVSDB2V31
MVSDB2V41
SQLRULES
DB2
STD
SQLWARN
NO
YES
STATICREADONLY
NO
YES

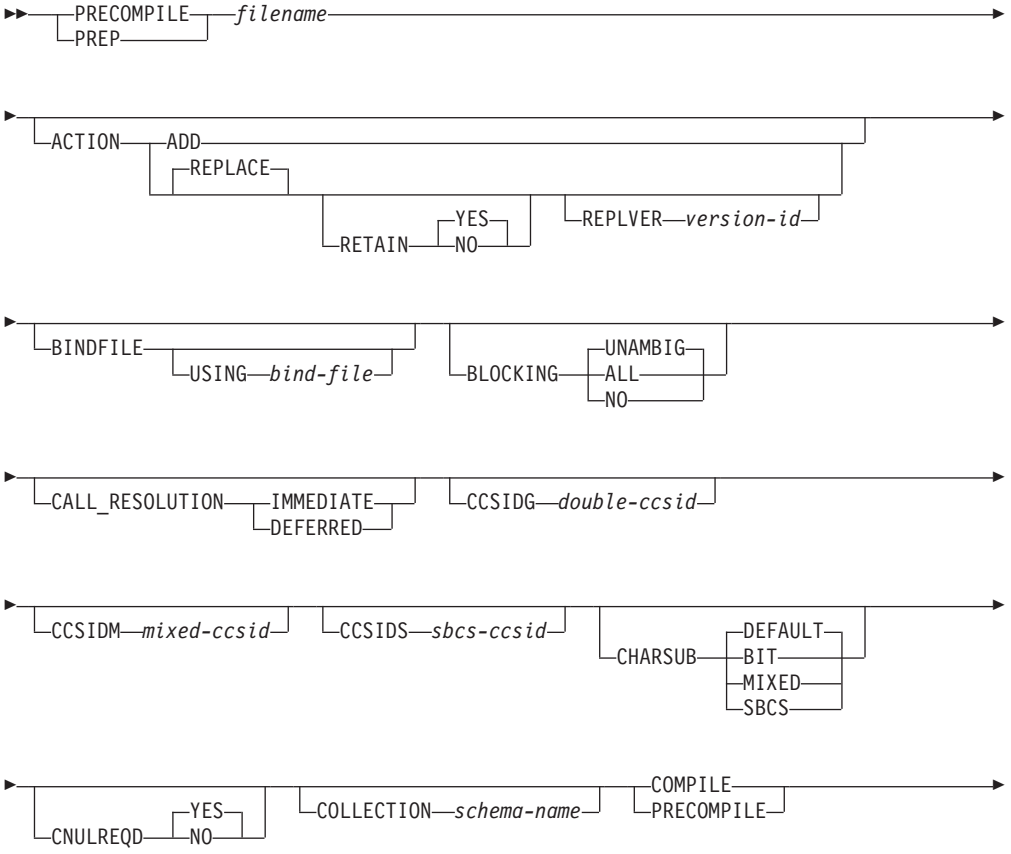


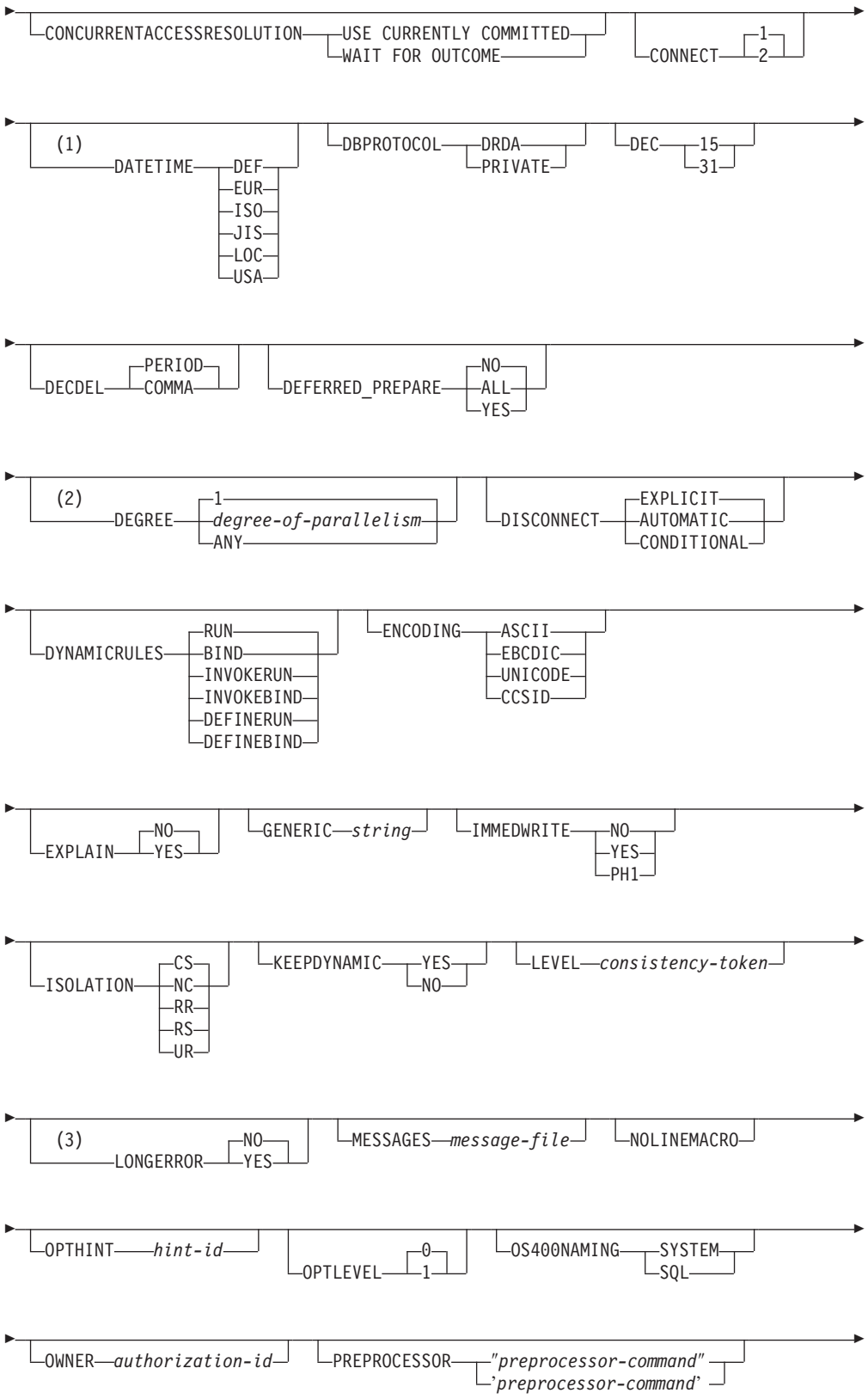
Notes:

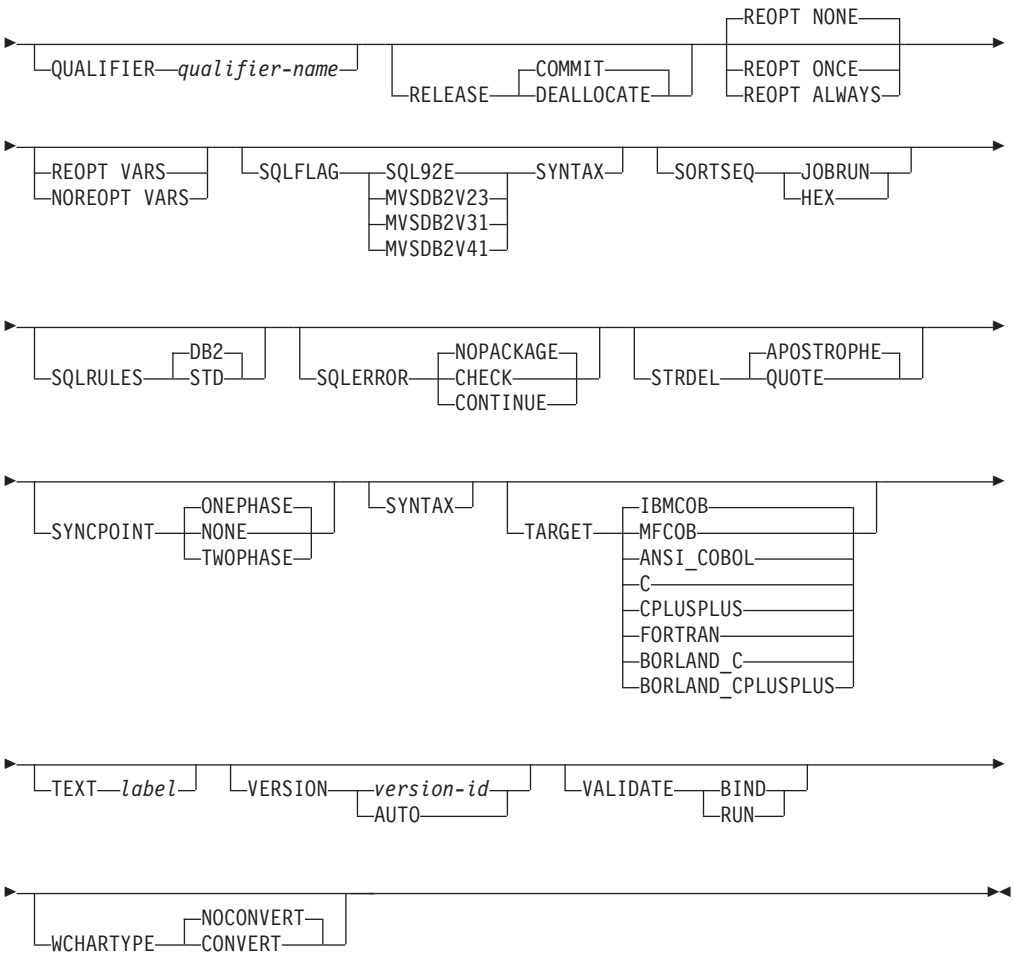
1 NO is the default for 32 bit systems and for 64 bit NT systems where long host variables can be used as declarations for INTEGER columns. YES is the default for 64 bit UNIX systems.

2 SYNTAX is a synonym for SQLERROR(CHECK).

For DB2 Database on servers other than Linux, Windows and UNIX




CONCURRENTACCESSRESOLUTION
USE CURRENTLY COMMITTED
WAIT FOR OUTCOME
CONNECT
1
2
(1)
DATETIME
DEF
EUR
ISO
JIS
LOC
USA
DBPROTOCOL
DRDA
PRIVATE
DEC
15
31
DECDEL
PERIOD
COMMA
DEFERRED_PREPARE
NO
ALL
YES
(2)
DEGREE
1
degree-of-parallelism
ANY
DISCONNECT
EXPLICIT
AUTOMATIC
CONDITIONAL
DYNAMICRULES
RUN
BIND
INVOKERUN
INVOKEBIND
DEFINERUN
DEFINEBIND
ENCODING
ASCII
EBCDIC
UNICODE
CCSID
EXPLAIN
NO
YES
GENERIC
string
IMMEDWRITE
NO
YES
PH1
ISOLATION
CS
NC
RR
RS
UR
KEEPDYNAMIC
YES
NO
LEVEL
consistency-token
(3)
LONGERROR
NO
YES
MESSAGES
message-file
NOLINEMACRO
OPTHINT
hint-id
OPTLEVEL
0
1
OS400NAMING
SYSTEM
SQL
OWNER
authorization-id
PREPROCESSOR
"preprocessor-command"
'preprocessor-command'



Notes:

1 If the server does not support the DATETIME DEF option, it is mapped to DATETIME ISO.

2 The DEGREE option is only supported by DRDA Level 2 Application Servers.

3 NO is the default for 32 bit systems and for 64 bit NT systems where long host variables can be used as declarations for INTEGER columns. YES is the default for 64 bit UNIX systems.

Command parameters

filename

Specifies the source file to be precompiled. An extension of:

- `.sqc` must be specified for C applications (generates a `.c` file)
- `.sqx` (Windows operating systems), or `.sqC` (UNIX and Linux operating systems) must be specified for C++ applications (generates a `.cxx` file on Windows operating systems, or a `.C` file on UNIX and Linux operating systems)
- `.sqb` must be specified for COBOL applications (generates a `.cbl` file)
- `.sqf` must be specified for FORTRAN applications (generates a `.for` file on Windows operating systems, or a `.f` file on UNIX and Linux operating systems).

The preferred extension for C++ applications containing embedded SQL on UNIX and Linux operating systems is `sqC`; however, the `sqx` convention, which was invented for systems that are not case sensitive, is tolerated by UNIX and Linux operating systems.

ACTION
Indicates whether the package can be added or replaced.

ADD Indicates that the named package does not exist, and that a new package is to be created. If the package already exists, execution stops, and a diagnostic error message is returned.

REPLACE
Indicates that the existing package is to be replaced by a new one with the same package name and creator. This is the default value for the ACTION option.

RETAIN
Indicates whether EXECUTE authorities are to be preserved when a package is replaced. If ownership of the package changes, the new owner grants the BIND and EXECUTE authority to the previous package owner.

NO Does not preserve EXECUTE authorities when a package is replaced. This value is not supported by DB2.

YES Preserves EXECUTE authorities when a package is replaced. This is the default value.

REPLVER *version-id*
Replaces a specific version of a package. The version identifier specifies which version of the package is to be replaced. If the specified version does not exist, an error is returned. If the REPLVER option of REPLACE is not specified, and a package already exists that matches the package name and version of the package being precompiled, that package will be replaced; if not, a new package will be added.

APREUSE
Specifies whether static SQL access plans are to be reused. When this option is enabled, the query compiler will attempt to reuse the access plans for the statement in any existing packages during the bind and during future implicit and explicit rebinds.

YES The query compiler will attempt to reuse the access plans for the statements in the package. If there is an existing package, the query compiler will attempt to reuse the access plan for every statement that can be matched with a statement in the new bind file. For a statement to match, the statement text must be identical and the section number for the statement in the existing package must match what the section number will be for the statement in the new package.

NO The query compiler will not attempt to reuse access plans for the statements in the package. This is the default setting.

BINDFILE
Results in the creation of a bind file. A package is not created unless the

package option is also specified. If a bind file is requested, but no package is to be created, as in the following example:

```
db2 prep sample.sqc bindfile
```

object existence and authorization SQLCODEs will be treated as warnings instead of errors. This will allow a bind file to be successfully created, even if the database being used for precompilation does not have all of the objects referred to in static SQL statements within the application. The bind file can be successfully bound, creating a package, once the required objects have been created.

USING *bind-file*
The name of the bind file that is to be generated by the precompiler. The file name must have an extension of `.bnd`. If a file name is not entered, the precompiler uses the name of the program (entered as the *filename* parameter), and adds the `.bnd` extension. If a path is not provided, the bind file is created in the current directory.

BLOCKING
Specifies the type of row blocking for cursors. The blocking of row data that contains references to LOB column data types is also supported in environments where the Database Partitioning Feature (DPF) is enabled.

ALL For cursors that are specified with the FOR READ ONLY clause or cursors not specified as FOR UPDATE, blocking occurs.

Ambiguous cursors are treated as read-only.

NO Blocking does not occur for any cursor.

For the definition of a read-only cursor and an ambiguous cursor, refer to *DECLARE CURSOR statement*.

Ambiguous cursors are treated as updatable.

UNAMBIG
For cursors that are specified with the FOR READ ONLY clause, blocking occurs.

Cursors that are not declared with the FOR READ ONLY or FOR UPDATE clause which are not ambiguous and are read-only will be blocked. Ambiguous cursors will not be blocked.

Ambiguous cursors are treated as updatable.

CALL_RESOLUTION
If set, the CALL_RESOLUTION DEFERRED option indicates that the CALL statement will be executed as an invocation of the deprecated sqleproc() API. If not set or if IMMEDIATE is set, the CALL statement will be executed as a normal SQL statement. SQL0204 will be issued if the precompiler fails to resolve the procedure on a CALL statement with CALL_RESOLUTION IMMEDIATE.

CCSIDG *double-ccsid*
An integer specifying the coded character set identifier (CCSID) to be used for double byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CCSIDM *mixed-ccsid*
An integer specifying the coded character set identifier (CCSID) to be used for mixed byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CCSIDS *sbcs-ccsid*
An integer specifying the coded character set identifier (CCSID) to be used for single byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CHARSUB
Designates the default character sub-type that is to be used for column definitions in CREATE and ALTER TABLE SQL statements. This DRDA precompile/bind option is not supported by DB2.

BIT Use the FOR BIT DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

DEFAULT
Use the target system defined default in all new character columns for which an explicit sub-type is not specified.

MIXED
Use the FOR MIXED DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

SBCS Use the FOR SBCS DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

CNULREQD
This option is related to the LANGLEVEL precompile option, which is not supported by DRDA. It is valid only if the bind file is created from a C or a C++ application. This DRDA bind option is not supported by DB2.

NO The application was coded on the basis of the LANGLEVEL SAA1 precompile option with respect to the null terminator in C string host variables.

YES The application was coded on the basis of the LANGLEVEL MIA precompile option with respect to the null terminator in C string host variables.

COLLECTION *schema-name*
Specifies a 128-byte collection identifier for the package. If not specified, the authorization identifier for the user processing the package is used.

CONCURRENTACCESSRESOLUTION
Specifies the concurrent access resolution to use for statements in the package.

USE CURRENTLY COMMITTED
Specifies that the database manager can use the currently committed version of the data for applicable scans when it is in the process of being updated or deleted. Rows in the process of being inserted can be skipped. This clause applies when the isolation level in effect is Cursor Stability or Read Stability (for Read

Stability it skips uncommited inserts only) and is ignored otherwise. Applicable scans include read-only scans that can be part of a read-only statement as well as a non read-only statement. The settings for the registry variables **DB2_EVALUNCOMMITTED**, **DB2_SKIPDELETED**, and **DB2_SKIPINSERTED** no longer apply.

WAIT FOR OUTCOME
Specifies Cursor Stability and higher scans to wait for the commit or rollback when encountering data in the process of being updated. Rows in the process of being inserted or deleted rows are not skipped. The settings for the registry variables **DB2_EVALUNCOMMITTED**, **DB2_SKIPDELETED**, and **DB2_SKIPINSERTED** no longer apply.

CONNECT

1 Specifies that a CONNECT statement is to be processed as a type 1 CONNECT.

2 Specifies that a CONNECT statement is to be processed as a type 2 CONNECT.

DATETIME
Specifies the date and time format to be used.

DEF Use a date and time format associated with the territory code of the database.

EUR Use the IBM standard for Europe date and time format.

ISO Use the date and time format of the International Standards Organization.

JIS Use the date and time format of the Japanese Industrial Standard.

LOC Use the date and time format in local form associated with the territory code of the database.

USA Use the IBM standard for U.S. date and time format.

DBPROTOCOL
Specifies what protocol to use when connecting to a remote site that is identified by a three-part name statement. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

DEC Specifies the maximum precision to be used in decimal arithmetic operations. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified.

15 15-digit precision is used in decimal arithmetic operations.

31 31-digit precision is used in decimal arithmetic operations.

DECDEL
Designates whether a period (.) or a comma (,) will be used as the decimal point indicator in decimal and floating point literals. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified.

COMMA
Use a comma (,) as the decimal point indicator.

PERIOD
Use a period (.) as the decimal point indicator.

DEFERRED_PREPARE
Provides a performance enhancement when accessing DB2 common server databases or DRDA databases. This option combines the SQL PREPARE statement flow with the associated OPEN, DESCRIBE, or EXECUTE statement flow to minimize inter-process or network flow.

NO The PREPARE statement will be executed at the time it is issued.

YES Execution of the PREPARE statement will be deferred until the corresponding OPEN, DESCRIBE, or EXECUTE statement is issued.

The PREPARE statement will not be deferred if it uses the INTO clause, which requires an SQLDA to be returned immediately. However, if the PREPARE INTO statement is issued for a cursor that does not use any parameter markers, the processing will be optimized by pre-OPENing the cursor when the PREPARE is executed.

ALL Same as YES, except that a PREPARE INTO statement is also deferred. If the PREPARE statement uses the INTO clause to return an SQLDA, the application must not reference the content of this SQLDA until the OPEN, DESCRIBE, or EXECUTE statement is issued and returned.

DEGREE
Specifies the degree of parallelism for the execution of static SQL statements in an SMP system. This option does not affect CREATE INDEX parallelism.

1 The execution of the statement will not use parallelism.

degree-of-parallelism
Specifies the degree of parallelism with which the statement can be executed, a value between 2 and 32 767 (inclusive).

ANY Specifies that the execution of the statement can involve parallelism using a degree determined by the database manager.

DISCONNECT

AUTOMATIC
Specifies that all database connections are to be disconnected at commit.

CONDITIONAL
Specifies that the database connections that have been marked RELEASE or have no open WITH HOLD cursors are to be disconnected at commit.

EXPLICIT
Specifies that only database connections that have been explicitly marked for release by the RELEASE statement are to be disconnected at commit.

DYNAMICRULES
Defines which rules apply to dynamic SQL at run time for the initial setting of the values used for authorization ID and for the implicit qualification of unqualified object references.

RUN Specifies that the authorization ID of the user executing the package is to be used for authorization checking of dynamic SQL statements. The authorization ID will also be used as the default package qualifier for implicit qualification of unqualified object references within dynamic SQL statements. This is the default value.

BIND Specifies that all of the rules that apply to static SQL for authorization and qualification are to be used at run time. That is, the authorization ID of the package owner is to be used for authorization checking of dynamic SQL statements, and the default package qualifier is to be used for implicit qualification of unqualified object references within dynamic SQL statements.

DEFINERUN
If the package is used within a routine context, the authorization ID of the routine definer is to be used for authorization checking and for implicit qualification of unqualified object references within dynamic SQL statements within the routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES RUN.

DEFINEBIND
If the package is used within a routine context, the authorization ID of the routine definer is to be used for authorization checking and for implicit qualification of unqualified object references within dynamic SQL statements within the routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES BIND.

INVOKERUN
If the package is used within a routine context, the current statement authorization ID in effect when the routine is invoked is to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES RUN.

INVOKEBIND
If the package is used within a routine context, the current statement authorization ID in effect when the routine is invoked is to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES BIND.

Because dynamic SQL statements will be using the authorization ID of the package owner in a package exhibiting bind behavior, the binder of the package should not have any authorities granted to them that the user of the package should not receive. Similarly, when defining a routine that will exhibit define behavior, the definer of the routine should not have any

authorities granted to them that the user of the package should not receive since a dynamic statement will be using the authorization ID of the routine's definer.

The following dynamically prepared SQL statements cannot be used within a package that was not bound with DYNAMICRULES RUN: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY, and SET EVENT MONITOR STATE.

ENCODING
Specifies the encoding for all host variables in static statements in the plan or package. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

EXPLAIN
Stores information in the Explain tables about the access plans chosen for each SQL statement in the package. DRDA does not support the ALL value for this option.

NO Explain information will not be captured.

YES Explain tables will be populated with information about the chosen access plan at prep/bind time for static statements and at run time for incremental bind statements.

If the package is to be used for a routine and the package contains incremental bind statements, then the routine must be defined as MODIFIES SQL DATA. If this is not done, incremental bind statements in the package will cause a run time error (SQLSTATE 42985).

REOPT
Explain information for each reoptimizable incremental bind SQL statement will be placed in the Explain tables at run time. In addition, Explain information will be gathered for reoptimizable dynamic SQL statements at run time, even if the CURRENT EXPLAIN MODE special register is set to NO.

If the package is to be used for a routine, then the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

ONLY The ONLY option allows you to explain statements without having the privilege to execute them. The explain tables are populated but no persistent package is created. If an existing package with the same name and version is encountered during the bind process, the existing package is neither dropped nor replaced even if you specified ACTION REPLACE. If an error occurs during population of the explain tables, explain information is not added for the statement that returned the error and for any statements that follow it.

ALL Explain information for each eligible static SQL statement will be placed in the Explain tables at prep/bind time. Explain information for each eligible incremental bind SQL statement will be placed in the Explain tables at run time. In addition, Explain information will be gathered for eligible dynamic SQL statements at run time, even if the CURRENT EXPLAIN MODE special register is set to NO.

If the package is to be used for a routine, then the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

EXPLSNAP
Stores Explain Snapshot information in the Explain tables. This DB2 precompile/bind option is not supported by DRDA.

NO An Explain Snapshot will not be captured.

YES An Explain Snapshot for each eligible static SQL statement will be placed in the Explain tables at prep/bind time for static statements and at run time for incremental bind statements.

If the package is to be used for a routine and the package contains incremental bind statements, then the routine must be defined as MODIFIES SQL DATA or incremental bind statements in the package will cause a run time error (SQLSTATE 42985).

REOPT
Explain Snapshot information for each reoptimizable incremental bind SQL statement will be placed in the Explain tables at run time. In addition, Explain Snapshot information will be gathered for reoptimizable dynamic SQL statements at run time, even if the CURRENT EXPLAIN SNAPSHOT special register is set to NO.

If the package is to be used for a routine, then the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

ALL An Explain Snapshot for each eligible static SQL statement will be placed in the Explain tables at prep/bind time. Explain Snapshot information for each eligible incremental bind SQL statement will be placed in the Explain tables at run time. In addition, Explain Snapshot information will be gathered for eligible dynamic SQL statements at run time, even if the CURRENT EXPLAIN SNAPSHOT special register is set to `NO`.

If the package is to be used for a routine, then the routine must be defined as MODIFIES SQL DATA, or incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

FEDERATED
Specifies whether a static SQL statement in a package references a nickname or a federated view. If this option is not specified and a static SQL statement in the package references a nickname or a federated view, a warning is returned and the package is created.

This option is not supported by DRDA servers.

NO A nickname or federated view is not referenced in the static SQL statements of the package. If a nickname or federated view is encountered in a static SQL statement during the prepare or bind phase of this package, an error is returned and the package is *not* created.

YES A nickname or federated view can be referenced in the static SQL statements of the package. If no nicknames or federated views are

encountered in static SQL statements during the prepare or bind of the package, no errors or warnings are returned and the package is created.

FEDERATED_ASYNCHRONY
Specifies the maximum number of asynchrony table queues (ATQs) that the federated server supports in the access plan for programs that use embedded SQL.

ANY 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.

number_of_atqs_in_the_plan
The number of ATQs in the plan. You specify a number in the range 0 to 32767.

FUNCPATH
Specifies the function path to be used in resolving user-defined distinct types and functions in static SQL. If this option is not specified, the default function path is ″SYSIBM″,″SYSFUN″,USER where USER is the value of the USER special register. This DB2 precompile/bind option is not supported by DRDA.

schema-name
An SQL identifier, either ordinary or delimited, which identifies a schema that exists at the application server. No validation that the schema exists is made at precompile or at bind time. The same schema cannot appear more than once in the function path. The schema name SYSPUBLIC cannot be specified for the function path. The number of schemas that can be specified is limited by the length of the resulting function path, which cannot exceed 2048 bytes. The schema SYSIBM does not need to be explicitly specified; it is implicitly assumed to be the first schema if it is not included in the function path.

INSERT
Allows a program being precompiled or bound against a DB2 Enterprise Server Edition server to request that data inserts be buffered to increase performance.

BUF Specifies that inserts from an application should be buffered.

DEF Specifies that inserts from an application should not be buffered.

GENERIC *string*
Supports the binding of new options that are defined in the target database, but are not supported by DRDA. Do not use this option to pass bind options that *are* defined in BIND or PRECOMPILE. This option can substantially improve dynamic SQL performance. The syntax is as follows:

```
generic "option1 value1 option2 value2 ..."
```

Each option and value must be separated by one or more blank spaces. For example, if the target DRDA database is DB2 Universal Database, Version 8, one could use:

```
generic "explsnap all queryopt 3 federated yes"
```

to bind each of the EXPLSNAP, QUERYOPT, and FEDERATED options.

The maximum length of the string is 32768 bytes.

IMMEDWRITE
Indicates whether immediate writes will be done for updates made to group buffer pool dependent pagesets or database partitions. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

ISOLATION
Determines how far a program bound to this package can be isolated from the effect of other executing programs.

CS Specifies Cursor Stability as the isolation level.

NC No Commit. Specifies that commitment control is not to be used. This isolation level is not supported by DB2.

RR Specifies Repeatable Read as the isolation level.

RS Specifies Read Stability as the isolation level. Read Stability ensures that the execution of SQL statements in the package is isolated from other application processes for rows read and changed by the application.

UR Specifies Uncommitted Read as the isolation level.

LANGLEVEL
Specifies the SQL rules that apply for both the syntax and the semantics for both static and dynamic SQL in the application. This option is not supported by DRDA servers.

MIA Select the ISO/ANS SQL92 rules as follows:

- To support error SQLCODE or SQLSTATE checking, an SQLCA must be declared in the application code.
- C null-terminated strings are padded with blanks and always include a null-terminating character, even if truncation occurs.
- The FOR UPDATE clause is optional for all columns to be updated in a positioned UPDATE.
- A searched UPDATE or DELETE requires SELECT privilege on the object table of the UPDATE or DELETE statement if a column of the object table is referenced in the search condition or on the right hand side of the assignment clause.
- A column function that can be resolved using an index (for example MIN or MAX) will also check for nulls and return warning SQLSTATE 01003 if there were any nulls.
- An error is returned when a duplicate unique constraint is included in a CREATE or ALTER TABLE statement.
- An error is returned when no privilege is granted and the grantor has no privileges on the object (otherwise a warning is returned).

SAA1 Select the common IBM DB2 rules as follows:

- To support error SQLCODE or SQLSTATE checking, an SQLCA must be declared in the application code.
- C null-terminated strings are not terminated with a null character if truncation occurs.
- The FOR UPDATE clause is required for all columns to be updated in a positioned UPDATE.

- A searched UPDATE or DELETE will not require SELECT privilege on the object table of the UPDATE or DELETE statement unless a fullselect in the statement references the object table.
- A column function that can be resolved using an index (for example MIN or MAX) will not check for nulls and warning SQLSTATE 01003 is not returned.
- A warning is returned and the duplicate unique constraint is ignored.
- An error is returned when no privilege is granted.

SQL92E

Defines the ISO/ANS SQL92 rules as follows:

- To support checking of SQLCODE or SQLSTATE values, variables by this name can be declared in the host variable declare section (if neither is declared, SQLCODE is assumed during precompilation).
- C null-terminated strings are padded with blanks and always include a null-terminating character, even if truncation occurs.
- The FOR UPDATE clause is optional for all columns to be updated in a positioned UPDATE.
- A searched UPDATE or DELETE requires SELECT privilege on the object table of the UPDATE or DELETE statement if a column of the object table is referenced in the search condition or on the right hand side of the assignment clause.
- A column function that can be resolved using an index (for example MIN or MAX) will also check for nulls and return warning SQLSTATE 01003 if there were any nulls.
- An error is returned when a duplicate unique constraint is included in a CREATE or ALTER TABLE statement.
- An error is returned when no privilege is granted and the grantor has no privileges on the object (otherwise a warning is returned).

KEEPDYNAMIC

Specifies whether dynamic SQL statements are to be kept after commit points. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

LEVEL *consistency-token*

Defines the level of a module using the consistency token. The consistency token is any alphanumeric value up to 8 characters in length. The RDB package consistency token verifies that the requester's application and the relational database package are synchronized. This option is not recommended for general use.

LONGERROR

Indicates whether long host variable declarations will be treated as an error. For portability, sqlint32 can be used as a declaration for an INTEGER column in precompiled C and C++ code.

NO Does not generate errors for the use of long host variable declarations. This is the default for 32 bit systems and for 64 bit NT systems where long host variables can be used as declarations

for INTEGER columns. The use of this option on 64 bit UNIX platforms will allow long host variables to be used as declarations for BIGINT columns.

YES Generates errors for the use of long host variable declarations. This is the default for 64 bit UNIX systems.

MESSAGES *message-file*
Specifies the destination for warning, error, and completion status messages. A message file is created whether the bind is successful or not. If a message file name is not specified, the messages are written to standard output. If the complete path to the file is not specified, the current directory is used. If the name of an existing file is specified, the contents of the file are overwritten.

NOLINEMACRO
Suppresses the generation of the #line macros in the output `.c` file. Useful when the file is used with development tools which require source line information such as profiles, cross-reference utilities, and debuggers. This precompile option is used for the C/C++ programming languages only.

OPTHINT
Controls whether query optimization hints are used for static SQL. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

OPTLEVEL
Indicates whether the C/C++ precompiler is to optimize initialization of internal SQLDAs when host variables are used in SQL statements. Such optimization can increase performance when a single SQL statement (such as FETCH) is used inside a tight loop.

0 Instructs the precompiler not to optimize SQLDA initialization.

1 Instructs the precompiler to optimize SQLDA initialization. This value should not be specified if the application uses:

- pointer host variables, as in the following example:

```
exec sql begin declare section;
char (*name)[20];
short *id;
exec sql end declare section;
```

- C++ data members directly in SQL statements.

OPTPROFILE *optimization-profile-name*
Specifies the name of an existing optimization profile to be used for all static statements in the package. The default value of the option is an empty string. The value also applies as the default for dynamic preparation of DML statements for which the CURRENT OPTIMIZATION PROFILE special register is null. If the specified name is unqualified, it is an SQL identifier, which is implicitly qualified by the QUALIFIER bind option.

The BIND command does not process the optimization file, but only validates that the name is syntactically valid. Therefore if the optimization profile does not exist or is invalid, an SQL0437W warning with reason code 13 will not occur until a DML statement is optimized using that optimization profile.

OUTPUT *filename*
Overrides the default name of the modified source file produced by the compiler. It can include a path.

OS400NAMING
Specifies which naming option is to be used when accessing DB2 for System i data. Supported by DB2 for System i only. For a list of supported option values, refer to the documentation for DB2 for System i.

Because of the slashes used as separators, a DB2 utility can still report a syntax error at execution time on certain SQL statements which use the System i system naming convention, even though the utility might have been precompiled or bound with the OS400NAMING SYSTEM option. For example, the Command Line Processor will report a syntax error on an SQL CALL statement if the System i system naming convention is used, whether or not it has been precompiled or bound using the OS400NAMING SYSTEM option.

OWNER *authorization-id*
Designates a 128-byte authorization identifier for the package owner. The owner must have the privileges required to execute the SQL statements contained in the package. Only a user with DBADM authority can specify an authorization identifier other than the user ID. The default value is the primary authorization ID of the precompile/bind process. SYSIBM, SYSCAT, and SYSSTAT are not valid values for this option. The *authorization-id* can only be a user (cannot be a role or a group).

PACKAGE
Creates a package. If neither PACKAGE, BINDFILE, nor SYNTAX is specified, a package is created in the database by default.

USING *package-name*
The name of the package that is to be generated by the precompiler. If a name is not entered, the name of the application program source file (minus extension and folded to uppercase) is used. Maximum length is 128 bytes.

PREPROCESSOR *"preprocessor-command"*

Specifies the preprocessor command that can be executed by the precompiler before it processes embedded SQL statements. The preprocessor command string (maximum length 1024 bytes) must be enclosed either by double or by single quotation marks.

This option enables the use of macros within the declare section. A valid preprocessor command is one that can be issued from the command line to invoke the preprocessor without specifying a source file. For example,

```
xlc -P -DMYMACRO=0
```

QUALIFIER *qualifier-name*
Provides an 128-byte implicit qualifier for unqualified objects contained in the package. The default is the owner's authorization ID, whether or not **owner** is explicitly specified.

QUERYOPT *optimization-level*
Indicates the desired level of optimization for all static SQL statements contained in the package. The default value is 5. The SET CURRENT QUERY OPTIMIZATION statement describes the complete range of optimization levels available. This DB2 precompile/bind option is not supported by DRDA.

RELEASE
Indicates whether resources are released at each COMMIT point, or when the application terminates. This DRDA precompile/bind option is not supported by DB2.

COMMIT
Release resources at each COMMIT point. Used for dynamic SQL statements.

DEALLOCATE
Release resources only when the application terminates.

REOPT
Specifies whether to have DB2 optimize an access path using values for host variables, parameter markers, global variables, and special registers. Valid values are:

NONE
The access path for a given SQL statement containing host variables, parameter markers, global variables, or special registers will not be optimized using real values for these variables. The default estimates for the these variables will be used instead, and this plan is cached and used subsequently. This is the default behavior.

ONCE The access path for a given SQL statement will be optimized using the real values of the host variables, parameter markers, global variables, or special registers when the query is first executed. This plan is cached and used subsequently.

ALWAYS
The access path for a given SQL statement will always be compiled and reoptimized using the values of the host variables, parameter markers, global variables, or special registers known at each execution time.

REOPT | NOREOPT VARS
These options have been replaced by REOPT ALWAYS and REOPT NONE; however, they are still supported for compatibility with previous releases. Specifies whether to have DB2 determine an access path at run time using values for host variables, global variables, parameter markers, and special registers. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

SQLCA
For FORTRAN applications only. This option is ignored if it is used with other languages.

NONE
Specifies that the modified source code is not consistent with the SAA definition.

SAA Specifies that the modified source code is consistent with the SAA definition.

SQLERROR
Indicates whether to create a package or a bind file if an error is encountered.

CHECK
Specifies that the target system performs all syntax and semantic checks on the SQL statements being bound. A package will not be created as part of this process. If, while binding, an existing package with the same name and version is encountered, the existing package is neither dropped nor replaced even if ACTION REPLACE was specified.

CONTINUE
Creates a package, even if errors occur when binding SQL statements. Those statements that failed to bind for authorization or existence reasons can be incrementally bound at execution time if VALIDATE RUN is also specified. Any attempt to execute them at run time generates an error (SQLCODE -525, SQLSTATE 51015).

NOPACKAGE
A package or a bind file is not created if an error is encountered.

SQLFLAG
Identifies and reports on deviations from the SQL language syntax specified in this option.

A bind file or a package is created only if the BINDFILE or the PACKAGE option is specified, in addition to the SQLFLAG option.

Local syntax checking is performed only if one of the following options is specified:

- BINDFILE
- PACKAGE
- SQLERROR CHECK
- SYNTAX

If SQLFLAG is not specified, the flagger function is not invoked, and the bind file or the package is not affected.

SQL92E SYNTAX
The SQL statements will be checked against ANSI or ISO SQL92 Entry level SQL language format and syntax with the exception of syntax rules that would require access to the database catalog. Any deviation is reported in the precompiler listing.

MVSDB2V23 SYNTAX
The SQL statements will be checked against MVS DB2 Version 2.3 SQL language syntax. Any deviation from the syntax is reported in the precompiler listing.

MVSDB2V31 SYNTAX
The SQL statements will be checked against MVS DB2 Version 3.1 SQL language syntax. Any deviation from the syntax is reported in the precompiler listing.

MVSDB2V41 SYNTAX
The SQL statements will be checked against MVS DB2 Version 4.1 SQL language syntax. Any deviation from the syntax is reported in the precompiler listing.

SORTSEQ
Specifies which sort sequence table to use on the System i system. Supported by DB2 for System i only. For a list of supported option values, refer to the documentation for DB2 for System i.

SQLRULES
Specifies:

- Whether type 2 CONNECTs are to be processed according to the DB2 rules or the Standard (STD) rules based on ISO/ANS SQL92.
- How an application specifies the format of LOB columns in the result set.

DB2

- Permits the SQL CONNECT statement to switch the current connection to another established (*dormant*) connection.
- This default setting allows an application to specify whether LOB values or LOB locators are retrieved only during the first fetch request. Subsequent fetch requests must use the same format for the LOB columns.

STD

- Permits the SQL CONNECT statement to establish a *new* connection only. The SQL SET CONNECTION statement must be used to switch to a dormant connection.
- The application can change between retrieving LOB values and LOB locators with each fetch request. This means that cursors with one or more LOB columns cannot be blocked, regardless of the BLOCKING bind option setting.

SQLWARN

Indicates whether warnings will be returned from the compilation of dynamic SQL statements (via PREPARE or EXECUTE IMMEDIATE), or from describe processing (via PREPARE...INTO or DESCRIBE).

NO Warnings will not be returned from the SQL compiler.

YES Warnings will be returned from the SQL compiler.

SQLCODE +238 is an exception. It is returned regardless of the **sqlwarn** option value.

STATICREADONLY

Determines whether static cursors will be treated as being READ ONLY. This DB2 precompile/bind option is not supported by DRDA.

NO All static cursors will take on the attributes as would normally be generated given the statement text and the setting of the LANGLEVEL precompile option. This is the default value.

YES Any static cursor that does not contain the FOR UPDATE or FOR READ ONLY clause will be considered READ ONLY.

STRDEL

Designates whether an apostrophe (') or double quotation marks (") will be used as the string delimiter within SQL statements. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified.

APOSTROPHE
Use an apostrophe (') as the string delimiter.

QUOTE
Use double quotation marks (") as the string delimiter.

SYNCPOINT

Specifies how commits or rollbacks are to be coordinated among multiple database connections. This command parameter is ignored and is only included here for backward compatibility.

NONE
Specifies that no Transaction Manager (TM) is to be used to perform a two-phase commit, and does not enforce single updater,

multiple reader. A COMMIT is sent to each participating database. The application is responsible for recovery if any of the commits fail.

ONEPHASE
Specifies that no TM is to be used to perform a two-phase commit. A one-phase commit is to be used to commit the work done by each database in multiple database transactions.

TWOPHASE
Specifies that the TM is required to coordinate two-phase commits among those databases that support this protocol.

SYNTAX
Suppresses the creation of a package or a bind file during precompilation. This option can be used to check the validity of the source file without modifying or altering existing packages or bind files. SYNTAX is a synonym for SQLERROR CHECK.

If SYNTAX is used together with the PACKAGE option, PACKAGE is ignored.

TARGET
Instructs the precompiler to produce modified code tailored to one of the supported compilers on the current platform.

IBMCOB
On AIX, code is generated for the IBM COBOL Set for AIX compiler.

MFCOB
Code is generated for the Micro Focus COBOL compiler. This is the default if a TARGET value is not specified with the COBOL precompiler on all Linux, UNIX and Windows operating systems.

ANSI_COBOL
Code compatible with the ANS X3.23-1985 standard is generated.

C Code compatible with the C compilers supported by DB2 on the current platform is generated.

CPLUSPLUS
Code compatible with the C++ compilers supported by DB2 on the current platform is generated.

FORTRAN
Code compatible with the FORTRAN compilers supported by DB2 on the current platform is generated.

TEXT *label*
The description of a package. Maximum length is 255 characters. The default value is blanks. This DRDA precompile/bind option is not supported by DB2.

TRANSFORM GROUP
Specifies the transform group name to be used by static SQL statements for exchanging user-defined structured type values with host programs. This transform group is not used for dynamic SQL statements or for the exchange of parameters and results with external functions or methods. This option is not supported by DRDA servers.

groupname
An SQL identifier of up to 128 bytes in length. A group name

cannot include a qualifier prefix and cannot begin with the prefix SYS since this is reserved for database use. In a static SQL statement that interacts with host variables, the name of the transform group to be used for exchanging values of a structured type is as follows:

- The group name in the TRANSFORM GROUP bind option, if any
- The group name in the TRANSFORM GROUP prep option as specified at the original precompilation time, if any
- The DB2_PROGRAM group, if a transform exists for the given type whose group name is DB2_PROGRAM
- No transform group is used if none of the above conditions exist.

The following errors are possible during the bind of a static SQL statement:

- SQLCODE yyy, SQLSTATE xxxxx: A transform is needed, but no static transform group has been selected.
- SQLCODE yyy, SQLSTATE xxxxx: The selected transform group does not include a necessary transform (TO SQL for input variables, FROM SQL for output variables) for the data type that needs to be exchanged.
- SQLCODE yyy, SQLSTATE xxxxx: The result type of the FROM SQL transform is not compatible with the type of the output variable, or the parameter type of the TO SQL transform is not compatible with the type of the input variable.

In these error messages, *yyyyy* is replaced by the SQL error code, and *xxxxx* by the SQL state code.

VALIDATE

Determines when the database manager checks for authorization errors and object not found errors. The package owner authorization ID is used for validity checking.

BIND Validation is performed at precompile/bind time. If all objects do not exist, or all authority is not held, error messages are produced. If SQLERROR CONTINUE is specified, a package/bind file is produced despite the error message, but the statements in error are not executable.

RUN Validation is attempted at bind time. If all objects exist, and all authority is held, no further checking is performed at execution time.

If all objects do not exist, or all authority is not held at precompile/bind time, warning messages are produced, and the package is successfully bound, regardless of the SQLERROR CONTINUE option setting. However, authority checking and existence checking for SQL statements that failed these checks during the precompile/bind process can be redone at execution time.

VERSION

Defines the version identifier for a package. If this option is not specified, the package version will be "" (the empty string).

version-id
Specifies a version identifier that is any alphanumeric value, $, #, @, _, -, or ., up to 64 characters in length.

AUTO
The version identifier will be generated from the consistency token. If the consistency token is a timestamp (it will be if the LEVEL option is not specified), the timestamp is converted into ISO character format and is used as the version identifier.

WCHARTYPE
Specifies the format for graphic data.

CONVERT
Host variables declared using the wchar_t base type will be treated as containing data in wchar_t format. Since this format is not directly compatible with the format of graphic data stored in the database (DBCS format), input data in wchar_t host variables is implicitly converted to DBCS format on behalf of the application, using the ANSI C function `wcstombs()`. Similarly, output DBCS data is implicitly converted to wchar_t format, using `mbstowcs()`, before being stored in host variables.

NOCONVERT
Host variables declared using the wchar_t base type will be treated as containing data in DBCS format. This is the format used within the database for graphic data; it is, however, different from the native wchar_t format implemented in the C language. Using NOCONVERT means that graphic data will not undergo conversion between the application and the database, which can improve efficiency. The application is, however, responsible for ensuring that data in wchar_t format is not passed to the database manager. When this option is used, wchar_t host variables should not be manipulated with the C wide character string functions, and should not be initialized with wide character literals (*L-literals*).

Usage notes

A modified source file is produced, which contains host language equivalents to the SQL statements. By default, a package is created in the database to which a connection has been established. The name of the package is the same as the file name (minus the extension and folded to uppercase), up to a maximum of 8 characters. Although the maximum length of a package name is 128 bytes, unless the PACKAGE USING option is specified, only the first 8 characters of the file name are used to maintain compatibility with previous versions of DB2.

Following connection to a database, PREP executes under the transaction that was started. PREP then issues a COMMIT or a ROLLBACK to terminate the current transaction and start another one.

Creating a package with a schema name that does not already exist results in the implicit creation of that schema. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.

During precompilation, an Explain Snapshot is not taken unless a package is created and EXPLSNAP has been specified. The snapshot is put into the Explain tables of the user creating the package. Similarly, Explain table information is only captured when EXPLAIN is specified, and a package is created.

Precompiling stops if a fatal error or more than 100 errors occur. If a fatal error occurs, the utility stops precompiling, attempts to close all files, and discards the package.

When a package exhibits bind behavior, the following will be true:

1. The implicit or explicit value of the BIND option OWNER will be used for authorization checking of dynamic SQL statements.
2. The implicit or explicit value of the BIND option QUALIFIER will be used as the implicit qualifier for qualification of unqualified objects within dynamic SQL statements.
3. The value of the special register CURRENT SCHEMA has no effect on qualification.

In the event that multiple packages are referenced during a single connection, all dynamic SQL statements prepared by those packages will exhibit the behavior as specified by the DYNAMICRULES option for that specific package and the environment they are used in.

If an SQL statement was found to be in error and the PRECOMPILE option SQLERROR CONTINUE was specified, the statement will be marked as invalid and another PRECOMPILE must be issued in order to change the state of the SQL statement. Implicit and explicit rebind will not change the state of an invalid statement in a package bound with VALIDATE RUN. A statement can change from static to incremental bind or incremental bind to static across implicit and explicit rebinds depending on whether or not object existence or authority problems exist during the rebind.

Binding a package with REOPT ONCE or REOPT ALWAYS might change static and dynamic statement compilation and performance.

For an embedded SQL program, if the FEDERATED_ASYNCHRONY precompile option is not explicitly specified the static statements in the package are bound using the FEDERATED_ASYNC configuration parameter. If the FEDERATED_ASYNCHRONY option is specified explicitly, that value is used for binding the packages and is also the initial value of the special register. Otherwise, the value of the database manager configuration parameter is used as the initial value of the special register. The FEDERATED_ASYNCHRONY precompile option influences dynamic SQL only when it is explicitly set.

REBIND

Allows the user to recreate a package stored in the database without the need for a bind file.

Authorization

One of the following:

- *dbadm* authority
- ALTERIN privilege on the schema
- BIND privilege on the package.

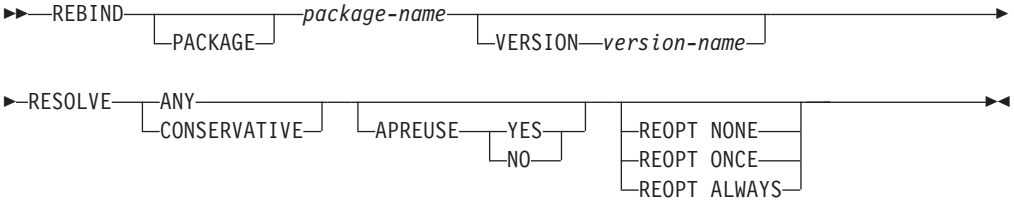
The authorization ID logged in the BOUNDBY column of the `SYSCAT.PACKAGES` system catalog table, which is the ID of the most recent binder of the package, is used as the binder authorization ID for the rebind, and for the default *schema* for

table references in the package. This default qualifier can be different from the authorization ID of the user executing the rebind request. REBIND will use the same bind options that were specified when the package was created.

Required connection

Database. If no database connection exists, and if implicit connect is enabled, a connection to the default database is made.

Command syntax



Command parameters

PACKAGE *package-name*
The qualified or unqualified name that designates the package to be rebound.

VERSION *version-name*
The specific version of the package to be rebound. When the version is not specified, it is taken to be "" (the empty string).

RESOLVE
Specifies whether rebinding of the package is to be performed with or without conservative binding semantics. This affects whether new objects that use the SQL path for resolution are considered during resolution on static DML statements in the package. This option is not supported by DRDA. Valid values are:

ANY All possible matches in the SQL path are considered for resolving references to any objects that use the SQL path for object resolution. Conservative binding semantics are not used. This is the default.

CONSERVATIVE
Only those objects in the SQL path that were defined before the last explicit bind time stamp are considered for resolving references to any objects that use the SQL path for object resolution. Conservative binding semantics are used. This option is not supported for an inoperative package.

APREUSE
Specifies whether static SQL access plans are to be reused. When this option is enabled, the query compiler will attempt to reuse the access plans for static SQL statements in the existing package during the rebind and during future implicit and explicit rebinds. The default is the value used during the previous invocation of the BIND or REBIND command or the ALTER PACKAGE statement. To determine the value, query the APREUSE column for the package in SYSCAT.PACKAGES.

YES The query compiler will attempt to reuse the access plans for the statements in the package.

NO The query compiler will not attempt to reuse access plans for the statements in the package.

REOPT

Specifies whether to have DB2 optimize an access path using values for host variables, parameter markers, global variables, and special registers.

NONE

The access path for a given SQL statement containing host variables, parameter markers, global variables, or special registers will not be optimized using real values for these variables. The default estimates for these variables will be used instead, and this plan is cached and used subsequently. This is the default behavior.

ONCE The access path for a given SQL statement will be optimized using the real values of the host variables, parameter markers, global variables, or special registers when the query is first executed. This plan is cached and used subsequently.

ALWAYS

The access path for a given SQL statement will always be compiled and re-optimized using the values of the host variables, parameter markers, global variables, or special registers known at each execution time.

Usage notes

REBIND does not automatically commit the transaction following a successful rebind. The user must explicitly commit the transaction. This enables "what if" analysis, in which the user updates certain statistics, and then tries to rebind the package to see what changes. It also permits multiple rebinds within a unit of work.

The REBIND command *will* commit the transaction if auto-commit is enabled.

This command:

- Provides a quick way to recreate a package. This enables the user to take advantage of a change in the system without a need for the original bind file. For example, if it is likely that a particular SQL statement can take advantage of a newly created index, the REBIND command can be used to recreate the package. REBIND can also be used to recreate packages after RUNSTATS has been executed, thereby taking advantage of the new statistics.
- Provides a method to recreate inoperative packages. Inoperative packages must be explicitly rebound by invoking either the bind utility or the rebind utility. A package will be marked inoperative (the VALID column of the `SYSCAT.PACKAGES` system catalog will be set to `X`) if a function instance on which the package depends is dropped.
- Gives users control over the rebinding of invalid packages. Invalid packages will be automatically (or implicitly) rebound by the database manager when they are executed. This might result in a noticeable delay in the execution of the first SQL request for the invalid package. It may be desirable to explicitly rebind invalid packages, rather than allow the system to automatically rebind them, in order to eliminate the initial delay and to prevent unexpected SQL error messages which might be returned in case the implicit rebind fails. For example, following database upgrade, all packages stored in the database will be invalidated by the UPGRADE DATABASE command. Given that this might involve a large number

of packages, it may be desirable to explicitly rebind all of the invalid packages at one time. This explicit rebinding can be accomplished using BIND, REBIND, or the db2rbind tool).

If multiple versions of a package (many versions with the same package name and creator) exist, only one version can be rebound at once. If not specified in the VERSION option, the package version defaults to be "". Even if there exists only one package with a name that matches, it will not be rebound unless its version matches the one specified or the default.

The choice of whether to use BIND or REBIND to explicitly rebind a package depends on the circumstances. It is recommended that REBIND be used whenever the situation does not specifically require the use of BIND, since the performance of REBIND is significantly better than that of BIND. BIND *must* be used, however:

- When there have been modifications to the program (for example, when SQL statements have been added or deleted, or when the package does not match the executable for the program).
- When the user wishes to modify any of the bind options as part of the rebind. REBIND does not support any bind options. For example, if the user wishes to have privileges on the package granted as part of the bind process, BIND must be used, since it has a GRANT option.
- When the package does not currently exist in the database.
- When detection of *all* bind errors is desired. REBIND only returns the first error it detects, whereas the BIND command returns the first 100 errors that occur during binding.

REBIND is supported by DB2 Connect.

If REBIND is executed on a package that is in use by another user, the rebind will not occur until the other user's logical unit of work ends, because an exclusive lock is held on the package's record in the `SYSCAT.PACKAGES` system catalog table during the rebind.

When REBIND is executed, the database manager recreates the package from the SQL statements stored in the `SYSCAT.STATEMENTS` system catalog table.

If REBIND encounters an error, processing stops, and an error message is returned.

REBIND will re-explain packages that were created with the EXPLSNAP bind option set to YES or ALL (indicated in the EXPLAIN_SNAPSHOT column in the `SYSCAT.PACKAGES` catalog table entry for the package) or with the EXPLAIN bind option set to YES or ALL (indicated in the EXPLAIN_MODE column in the `SYSCAT.PACKAGES` catalog table entry for the package). The Explain tables used are those of the REBIND requester, not the original binder.

If an SQL statement was found to be in error and the BIND option SQLERROR CONTINUE was specified, the statement will be marked as invalid even if the problem has been corrected. REBIND will not change the state of an invalid statement. In a package bound with VALIDATE RUN, a statement can change from static to incremental bind or incremental bind to static across a REBIND depending on whether or not object existence or authority problems exist during the REBIND.

Rebinding a package with REOPT ONCE | ALWAYS might change static and dynamic statement compilation and performance.

If REOPT is not specified, REBIND will preserve the existing REOPT value used at precompile or bind time.

Chapter 31. Application programming interfaces (APIs)

Following are the application programming interfaces (APIs) that correspond to the DB2 commands that are used for the Common Criteria evaluation.

DB2 UDB APIs for Administrators

db2Backup - Back up a database or table space

Creates a backup copy of a database or a table space.

Scope

In a partitioned database environment, by default this API affects only the database partition on which it is executed.

If the option to perform a partitioned backup is specified, the command can be called only on the catalog node. If the option specifies that all database partition servers are to be backed up, it affects all database partition servers that are listed in the `db2nodes.cfg` file. Otherwise, it affects the database partition servers that are specified on the API.

Authorization

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection

Database. This API automatically establishes a connection to the specified database.

The connection will be terminated upon the completion of the backup.

API include file

`db2ApiDf.h`

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Backup (
       db2Uint32 versionNumber,
       void * pDB2BackupStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2BackupStruct
{
  char *piDBAlias;
  char oApplicationId[SQLU_APPLID_LEN+1];
  char oTimestamp[SQLU_TIME_STAMP_LEN+1];
  struct db2TablespaceStruct *piTablespaceList;
  struct db2MediaListStruct *piMediaList;
  char *piUsername;
  char *piPassword;
```

```
  void *piVendorOptions;
  db2Uint32 iVendorOptionsSize;
  db2Uint32 oBackupSize;
  db2Uint32 iCallerAction;
  db2Uint32 iBufferSize;
  db2Uint32 iNumBuffers;
  db2Uint32 iParallelism;
  db2Uint32 iOptions;
  db2Uint32 iUtilImpactPriority;
  char *piComprLibrary;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  db2NodeType *piNodeList;
  db2int32 iNumMPPOutputStructs;
  struct db2BackupMPPOutputStruct *poMPPOutputStruct;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                        **tablespaces;
  db2Uint32 numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                        **locations;
  db2Uint32 numLocations;
  char locationType;
} db2MediaListStruct;

typedef SQL_STRUCTURE db2BackupMPPOutputStruct
{
  db2NodeType nodeNumber;
  db2Uint64 backupSize;
  struct sqlca  sqlca;
} db2BackupMPPOutputStruct;

SQL_API_RC SQL_API_FN
  db2gBackup (
       db2Uint32 versionNumber,
       void * pDB2gBackupStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gBackupStruct
{
  char *piDBAlias;
  db2Uint32 iDBAliasLen;
  char *poApplicationId;
  db2Uint32 iApplicationIdLen;
  char *poTimestamp;
  db2Uint32 iTimestampLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct *piMediaList;
  char *piUsername;
  db2Uint32 iUsernameLen;
  char *piPassword;
  db2Uint32 iPasswordLen;
  void *piVendorOptions;
  db2Uint32 iVendorOptionsSize;
  db2Uint32 oBackupSize;
  db2Uint32 iCallerAction;
  db2Uint32 iBufferSize;
  db2Uint32 iNumBuffers;
  db2Uint32 iParallelism;
  db2Uint32 iOptions;
```

```
  db2Uint32 iUtilImpactPriority;
  char *piComprLibrary;
  db2Uint32 iComprLibraryLen;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  db2NodeType *piNodeList;
  db2int32 iNumMPPOutputStructs;
  struct db2gBackupMPPOutputStruct *poMPPOutputStruct;
} db2gBackupStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char *tablespaces;
  db2Uint32 numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char *locations;
  db2Uint32 numLocations;
  char locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2gBackupMPPOutputStruct
{
  db2NodeType nodeNumber;
  db2Uint64 backupSize;
  struct sqlca  sqlca;
} db2gBackupMPPOutputStruct;

typedef SQL_STRUCTURE db2Char
{
   char *pioData;
   db2Uint32 iLength;
   db2Uint32 oLength;
} db2Char;
```

db2Backup API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter **pDB2BackupStruct**.

pDB2BackupStruct
Input. A pointer to the db2BackupStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2BackupStruct data structure parameters

piDBAlias
Input. A string containing the database alias (as cataloged in the system database directory) of the database to back up.

oApplicationId
Output. The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

oTimestamp
Output. The API will return the time stamp of the backup image

piTablespaceList
Input. List of table spaces to be backed up. Required for table space level backup only. Must be NULL for a database level backup. See structure db2TablespaceStruct.

piMediaList
Input. This structure allows the caller to specify the destination for the backup operation. For more information, see the db2MediaListStruct structure below.

piUsername
Input. A string containing the user name to be used when attempting a connection. Can be NULL.

piPassword
Input. A string containing the password to be used with the user name. Can be NULL.

piVendorOptions
Input. Used to pass information from the application to the vendor functions. This data structure must be flat; that is, no level of indirection is supported. Note that byte-reversal is not done, and code page is not checked for this data.

iVendorOptionsSize
Input. The length of the **piVendorOptions** field, which cannot exceed 65535 bytes.

oBackupSize
Output. Size of the backup image (in MB).

iCallerAction
Input. Specifies action to be taken. Valid values (defined in `db2ApiDf` header file, located in the include directory) are:

DB2BACKUP_BACKUP
Start the backup.

DB2BACKUP_NOINTERRUPT
Start the backup. Specifies that the backup will run unattended, and that scenarios which normally require user intervention will either be attempted without first returning to the caller, or will generate an error. Use this caller action, for example, if it is known that all of the media required for the backup have been mounted, and utility prompts are not desired.

DB2BACKUP_CONTINUE
Continue the backup after the user has performed some action requested by the utility (mount a new tape, for example).

DB2BACKUP_TERMINATE
Terminate the backup after the user has failed to perform some action requested by the utility.

DB2BACKUP_DEVICE_TERMINATE
Remove a particular device from the list of devices used by backup. When a particular medium is full, backup will return a warning to the caller (while continuing to process using the remaining devices). Call backup again with this caller action to remove the device which generated the warning from the list of devices being used.

DB2BACKUP_PARM_CHK
Used to validate parameters without performing a backup. This option does not terminate the database connection after the call returns. After successful return of this call, it is expected that the user will issue a call with SQLUB_CONTINUE to proceed with the action.

DB2BACKUP_PARM_CHK_ONLY
Used to validate parameters without performing a backup. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.

iBufferSize
Input. Backup buffer size in 4 KB allocation units (pages). Minimum is 8 units.

iNumBuffers
Input. Specifies number of backup buffers to be used. Minimum is 2. Maximum is limited by memory.

iParallelism
Input. Degree of parallelism (number of buffer manipulators). Minimum is 1. Maximum is 1024.

iOptions
Input. A bitmap of backup properties. The options are to be combined using the bitwise OR operator to produce a value for **iOptions**. Valid values (defined in `db2ApiDf` header file, located in the include directory) are:

DB2BACKUP_OFFLINE
Offline gives an exclusive connection to the database.

DB2BACKUP_ONLINE
Online allows database access by other applications while the backup operation occurs.

Note: An online backup operation may appear to hang if users are holding locks on SMS LOB data.

DB2BACKUP_DB
Full database backup.

DB2BACKUP_TABLESPACE
Table space level backup. For a table space level backup, provide a list of table spaces in the **piTablespaceList** parameter.

DB2BACKUP_INCREMENTAL
Specifies a cumulative (incremental) backup image. An incremental backup image is a copy of all database data that has changed since the most recent successful, full backup operation.

DB2BACKUP_DELTA
Specifies a noncumulative (delta) backup image. A delta backup image is a copy of all database data that has changed since the most recent successful backup operation of any type.

DB2BACKUP_COMPRESS
Specifies that the backup should be compressed.

DB2BACKUP_INCLUDE_COMPR_LIB
Specifies that the library used for compressing the backup should be included in the backup image.

DB2BACKUP_EXCLUDE_COMPR_LIB
Specifies that the library used for compressing the backup should be not included in the backup image.

DB2BACKUP_INCLUDE_LOGS
Specifies that the backup image should also include the range of log files required to restore and roll forward this image to some consistent point in time. This option is not valid for an offline backup or a multi-partition backup.

DB2BACKUP_EXCLUDE_LOGS
Specifies that the backup image should not include any log files.

Note: When performing an offline backup operation, logs are excluded whether or not this option is specified, with the exception of snapshot backups where INCLUDE is the default.

DB2BACKUP_MPP
Perform backup in a manner suitable for a partitioned database.

iUtilImpactPriority
Input. Specifies the priority value to be used during a backup.

- If this value is non-zero, the utility will run throttled. Otherwise, the utility will run unthrottled.
- If there are multiple concurrent utilities running, this parameter is used to determine a relative priority between the throttled tasks. For example, consider two concurrent backups, one with priority 2 and another with priority 4. Both will be throttled, but the one with priority 4 will be allotted more resources. Setting priorities to 2 and 4 is no different than setting them to 5 and 10 or 30 and 60. Priorities values are purely relative.

piComprLibrary
Input. Indicates the name of the external library to be used to perform compression of the backup image. The name must be a fully-qualified path referring to a file on the server. If the value is a null pointer or a pointer to an empty string, DB2 will use the default library for compression. If the specified library is not found, the backup will fail.

piComprOptions
Input. Describes a block of binary data that will be passed to the initialization routine in the compression library. DB2 will pass this string directly from the client to the server, so any issues of byte-reversal or code-page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of **piComprOptions** and **iComprOptionsSize** with the contents and size of this file respectively and will pass these new values to the initialization routine instead.

iComprOptionsSize
Input. A four-byte unsigned integer representing the size of the block of data passed as **piComprOptions**. **iComprOptionsSize** shall be zero if and only if **piComprOptions** is a null pointer.

iAllNodeFlag
Input. Partitioned database environments only. Indicates whether the backup operation is to be applied to all or some database partition servers defined in `db2nodes.cfg`. Valid values are:

DB2_NODE_LIST
Apply to database partition servers in a list that is passed in **piNodeList**.

DB2_ALL_NODES
Apply to all database partition servers. **piNodeList** should be NULL. This is the default value.

DB2_ALL_EXCEPT
Apply to all database partition servers except those in a list that is passed in **piNodeList**.

iNumNodes
Input. Specifies the number of database partition servers in the **piNodeList** array.

piNodeList
Input. A pointer to an array of database partition server numbers on which to perform the backup.

iNumMPPOutputStructs
Input. Specifies the number of elements in the **piMPPOutputStruct** array. This must be equal to or greater than the number of database partition servers that participate in this backup operation.

piMPPOutputStruct
Output. A pointer to an array of db2BackupMPPOutputStruct structures that specify output parameters for particular database partition servers.

db2TablespaceStruct data structure specific parameters

tablespaces
Input. A pointer to the list of table spaces to be backed up. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numTablespaces
Input. Number of entries in the **tablespaces** parameter.

db2MediaListStruct data structure parameters

locations
Input. A pointer to the list of media locations. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numLocations
Input. The number of entries in the locations parameter.

locationType
Input. A character indicating the media type. Valid values (defined in `sqlutil` header file, located in the include directory.) are:

SQLU_LOCAL_MEDIA: 'L'
Local devices (tapes, disks, diskettes, or named pipes).

SQLU_XBSA_MEDIA: 'X'
XBSA interface.

SQLU_TSM_MEDIA: 'A'
Tivoli Storage Manager.

SQLU_OTHER_MEDIA: 'O'
Vendor library.

SQLU_SNAPSHOT_MEDIA: 'F'
Specifies that a snapshot backup is to be taken.

You cannot use SQLU_SNAPSHOT_MEDIA with any of the following:

- DB2BACKUP_COMPRESS
- DB2BACKUP_TABLESPACE
- DB2BACKUP_INCREMENTAL
- **iNumBuffers**
- **iBufferSize**
- **iParallelism**
- **piComprOptions**
- **iUtilImpactPriority**
- **numLocations** field of this structure must be 1 for snapshot restore

The default behavior for a snapshot backup is a FULL DATABASE OFFLINE backup of all paths that make up the database including all containers, local volume directory, database path (DBPATH), and primary log and mirror log paths (INCLUDE LOGS is the default for all snapshot backups unless EXCLUDE LOGS is explicitly stated).

Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage SAN Volume Controller
- IBM Enterprise Storage Server Model 800
- IBM System Storage DS6000
- IBM System Storage DS8000
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS

db2BackupMPPOutputStruct and db2gBackupMPPOutputStruct data structure parameters

nodeNumber
The database partition server to which the option applies.

backupSize
The size of the backup on the specified database partition, in kilobytes.

sqlca The sqlca from the specified database partition.

db2gBackupStruct data structure specific parameters

iDBAliasLen
Input. A 4-byte unsigned integer representing the length in bytes of the database alias.

iApplicationIdLen
Input. A 4-byte unsigned integer representing the length in bytes of the **poApplicationId** buffer. Should be equal to SQLU_APPLID_LEN+1 (defined in `sqlutil.h`).

iTimestampLen
Input. A 4-byte unsigned integer representing the length in bytes of the **poTimestamp** buffer. Should be equal to SQLU_TIME_STAMP_LEN+1 (defined in `sqlutil.h`).

iUsernameLen
Input. A 4-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is provided.

iPasswordLen
Input. A 4-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

iComprLibraryLen
Input. A four-byte unsigned integer representing the length in bytes of the name of the library specified in **piComprLibrary**. Set to zero if no library name is given.

db2Char data structure parameters

pioData
A pointer to a character data buffer. If NULL, no data will be returned.

iLength
Input. The size of the **pioData** buffer.

oLength
Output. The number of valid characters of data in the **pioData** buffer.

Usage notes

You can only perform a snapshot backup with **versionNumber** db2Version950 or higher. If you specify media type SQLU_SNAPSHOT_MEDIA with a **versionNumber** lower than db2Version950, DB2 database will return an error.

This function is exempt from all label-based access control (LBAC) rules. It backs up all data, even protected data. Also, the data in the backup itself is not protected by LBAC. Any user with the backup and a place in which to restore it can gain access to the data.

As you regularly backup your database, you might accumulate very large database backup images, many database logs and load copy images, all of which might be taking up a large amount of disk space. Refer to "Managing recovery objects" for information on how to manage these recovery objects.

Usage notes for a single system view (SSV) backup in a partitioned database environment

- To perform an SSV backup, specify **iOptions** DB2BACKUP_MPP and one of DB2BACKUP_DB or DB2BACKUP_TABLESPACE.
- You can only perform a SSV backup with **versionNumber** db2Version950 or higher. If you specify **iOptions** DB2BACKUP_MPP with a **versionNumber** lower than db2Version950, DB2 database will

return an error. If you specify other options related to SSV backup with a **versionNumber** lower than db2Version950, DB2 database will ignore those options.

- The values for **piMediaList** specified directly in db2BackupStruct will be used as the default values on all nodes.
- The value of **oBackupSize** returned in the db2BackupStruct is the sum of the backup sizes on all nodes. The value of **backupSize** returned in the db2BackupMPPOutputStruct is the size of the backup on the specified database partition.
- **iAllNodeFlag**, **iNumNodes**, and **piNodeList** operate the same as the similarly-named elements in db2RollforwardStruct, with the exception that there is no CAT_NODE_ONLY value for **iAllNodeFlag**.
- SSV backups performed with the DB2BACKUP_BACKUP caller action are performed as if the DB2BACKUP_NOINTERRUPT caller action was specified.
- ***poMPPOutputStruct** points to memory allocated by the caller that contains at least as many elements as there are database partitions participating in the backup.

db2CfgGet - Get the database manager or database configuration parameters

Returns the values of individual entries in a specific database configuration file or a database manager configuration file.

Scope

Information about a specific database configuration file is returned only for the database partition on which it is executed.

Authorization

None

Required connection

To obtain the current online value of a configuration parameter for a specific database configuration file, a connection to the database is required. To obtain the current online value of a configuration parameter for the database manager, an instance attachment is required. Otherwise, a connection to a database or an attachment to an instance is not required.

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2CfgGet (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2Cfg
{
   db2Uint32 numItems;
```

```
   struct db2CfgParam *paramArray;
   db2Uint32 flags;
   char *dbname;
} db2Cfg;

typedef SQL_STRUCTURE db2CfgParam
{
   db2Uint32 token;
   char *ptrvalue;
   db2Uint32 flags;
} db2CfgParam;

SQL_API_RC SQL_API_FN
  db2gCfgGet (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gCfg
{
   db2Uint32 numItems;
   struct db2gCfgParam *paramArray;
   db2Uint32 flags;
   db2Uint32 dbname_len;
   char *dbname;
} db2gCfg;

typedef SQL_STRUCTURE db2gCfgParam
{
   db2Uint32 token;
   db2Uint32 ptrvalue_len;
   char *ptrvalue;
   db2Uint32 flags;
} db2gCfgParam;
```

db2CfgGet API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2Cfg structure.

pSqlca
Output. A pointer to the sqlca structure.

db2Cfg data structure parameters

numItems
Input. The number of configuration parameters in the paramArray array. Set this value to db2CfgMaxParam to specify the largest number of elements in the paramArray.

paramArray
Input. A pointer to the db2CfgParam structure.

flags Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

db2CfgDatabase
Specifies to return the values in the database configuration file.

db2CfgDatabaseManager
Specifies to return the values in the database manager configuration file.

db2CfgImmediate
Returns the current values of the configuration parameters stored in memory.

db2CfgDelayed
Gets the values of the configuration parameters on disk. These do not become the current values in memory until the next database connection or instance attachment.

db2CfgGetDefaults
Returns the default values for the configuration parameter.

db2CfgReset
Reset to default values.

dbname
Input. The database name.

db2CfgParam data structure parameters

token Input. The configuration parameter identifier.

Valid entries and data types for the db2CfgParam token element are listed in ″Configuration parameters summary″.

ptrvalue
Output. The configuration parameter value.

flags Output. Provides specific information for each parameter in a request. Valid values (defined in `db2ApiDf` header file, located in the include directory) are:

db2CfgParamAutomatic
Indicates whether the retrieved parameter has a value of automatic. To determine whether a given configuration parameter has been set to automatic, perform a boolean AND operation against the value returned by the flag and the db2CfgParamAutomatic keyword defined in `db2ApiDf.h`.

db2CfgParamComputed
Indicates whether the retrieved parameter has a value of computed. To determine whether a given configuration parameter has been set to computed, perform a boolean AND operation against the value returned by the flag and the db2CfgParamComputed keyword defined in `db2ApiDf.h`.

If the boolean AND operation is false for both of the keywords above, it means that the retrieved parameter value is set manually.

db2gCfg data structure specific parameters

dbname_len
Input. The length in bytes of dbname parameter.

db2gCfgParam data structure specific parameters

ptrvalue_len
Input. The length in bytes of ptrvalue parameter.

Usage notes

The configuration parameters `maxagents` and `maxcagents` are deprecated. In a future release, these configuration parameters may be removed completely.

The db2CfgGet API will tolerate requests for SQLF_KTN_MAXAGENTS and SQLF_KTN_MAXCAGENTS, but 0 will be returned if the server is DB2 v9.5.

db2CfgSet - Set the database manager or database configuration parameters

Modifies individual entries in a specific database configuration file or a database manager configuration file. A database configuration file resides on every node on which the database has been created.

Scope

Modifications to the database configuration file affect all database partitions by default.

Authorization

For modifications to the database configuration file, one of the following:

- sysadm
- sysctrl
- sysmaint

For modifications to the database manager configuration file:

- sysadm

Required connection

To make an online modification of a configuration parameter for a specific database, a connection to the database is required. To make an online modification of a configuration parameter for the database manager, an attachment to an instance is not required.

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
   db2CfgSet (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2Cfg
{
   db2Uint32 numItems;
   struct db2CfgParam *paramArray;
   db2Uint32 flags;
   char *dbname;
   SQL_PDB_NODE_TYPE dbpartitionnum;
}  db2Cfg;
```

```
typedef SQL_STRUCTURE db2CfgParam
{
   db2Uint32 token;
   char *ptrvalue;
   db2Uint32 flags;
}  db2CfgParam;

SQL_API_RC SQL_API_FN
   db2gCfgSet (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gCfg
{
   db2Uint32 numItems;
   struct db2gCfgParam *paramArray;
   db2Uint32 flags;
   db2Uint32 dbname_len;
   char *dbname;
}  db2gCfg;

typedef SQL_STRUCTURE db2gCfgParam
{
   db2Uint32 token;
   db2Uint32 ptrvalue_len;
   char *ptrvalue;
   db2Uint32 flags;
}  db2gCfgParam;
```

db2CfgSet API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2Cfg structure.

pSqlca
Output. A pointer to the sqlca structure.

db2Cfg data structure parameters

numItems
Input. The number of configuration parameters in the paramArray array. Set this value to db2CfgMaxParam to specify the largest number of elements in the paramArray.

paramArray
Input. A pointer to the db2CfgParam structure.

flags Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

db2CfgDatabase
Specifies to return the values in the database configuration file.

db2CfgDatabaseManager
Specifies to return the values in the database manager configuration file.

db2CfgImmediate
Returns the current values of the configuration parameters stored in memory.

db2CfgDelayed
Gets the values of the configuration parameters on disk. These do not become the current values in memory until the next database connection or instance attachment.

db2CfgGetDefaults
Returns the default values for the configuration parameter.

db2CfgReset
Reset to default values.

db2CfgSingleDbpartition
To update or reset the database configuration on a specific database partition, set this flag and provide a value for dbpartitionnum.

dbname
Input. The database name.

dbpartitionnum
Input. Specifies on which database partition this API will set the configuration value.

db2CfgParam data structure parameters

token Input. The configuration parameter identifier. Valid entries and data types for the db2CfgParam token element are listed in ″Configuration parameters summary″.

ptrvalue
Output. The configuration parameter value.

flags Input. Provides specific information for each parameter in a request. Valid values (defined in `db2ApiDf` header file, located in the include directory) are:

db2CfgParamAutomatic
Indicates whether the retrieved parameter has a value of automatic. To determine whether a given configuration parameter has been set to automatic, perform a boolean AND operation against the value returned by the flag and the db2CfgParamAutomatic keyword defined in db2ApiDf.h.

db2CfgParamComputed
Indicates whether the retrieved parameter has a value of computed. To determine whether a given configuration parameter has been set to computed, perform a boolean AND operation against the value returned by the flag and the db2CfgParamComputed keyword defined in `db2ApiDf.h`.

db2CfgParamManual
Used to unset the automatic or computed setting of a parameter and set the parameter to the current value. The **ptrvalue** field is ignored and can be set to NULL.

db2gCfg data structure specific parameters

dbname_len
Input. The length in bytes of dbname parameter.

db2gCfgParam data structure specific parameters

ptrvalue_len
Input. The length in bytes of ptrvalue parameter.

Usage notes

The configuration parameters `maxagents` and `maxcagents` are deprecated. In a future release, these configuration parameters may be removed completely.

The db2CfgSet API will tolerate updates to the `maxagents` and `maxcagents` configuration parameters, however these updates will be ignored by DB2.

Usage samples

CASE 1: The MAXAPPLS parameter will be set to 50 at dbpartitionnum 30.

CASE 2: The MAXAPPLS parameter will be set to 50 on all dbpartitionnum.

```
int updateDbConfig()
{
    struct sqlca sqlca = {0};
    db2Cfg cfgStruct = {0};
    db2CfgParam cfgParameters[2];
    char *dbAlias = "SAMPLE";

    /* initialize cfgParameters */
    cfgParameters[0].flags = 0;
    cfgParameters[0].token = SQLF_DBTN_TSM_OWNER;
    cfgParameters[0].ptrvalue = (char *)malloc(sizeof(char) * 65);
    cfgParameters[1].flags = 0;
    cfgParameters[1].token = SQLF_DBTN_MAXAPPLS;
    cfgParameters[1].ptrvalue = (char *)malloc(sizeof(sqluint16));

    /* set two DB Config. fields  */
    strcpy(cfgParameters[0].ptrvalue, "tsm_owner");
    *(sqluint16 *)(cfgParameters[1].ptrvalue) = 50;

    /* initialize cfgStruct to update db cfg on single partition*/
    cfgStruct.numItems = 2;
    cfgStruct.paramArray = cfgParameters;
    cfgStruct.flags = db2CfgDatabase | db2CfgImmediate;
    cfgStruct.flags |= db2CfgSingleDbpartition;
    cfgStruct.dbname = dbAlias;
    cfgStruct.dbpartitionnum = 30;

    /* CASE 1: update database configuration */
    db2CfgSet(db2Version950, (void *)&cfgStruct, &sqlca);

    /* set cfgStruct to update db cfg on all db partitions */
    cfgStruct.flags &= ~db2CfgSingleDbpartition;

    /* CASE 2: update database configuration */
    db2CfgSet(db2Version950, (void *)&cfgStruct, &sqlca);
    ..............
}
```

db2DatabaseRestart - Restart database

Restarts a database that has been abnormally terminated and left in an inconsistent state. At the successful completion of this API, the application remains connected to the database if the user has CONNECT privilege.

Scope

This API affects only the database partition server on which it is executed.

Authorization

None

Required connection

This API establishes a database connection.

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2DatabaseRestart (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef struct db2RestartDbStruct
{
   char *piDatabaseName;
   char *piUserId;
   char *piPassword;
   char *piTablespaceNames;
   db2int32 iOption;
} db2RestartDbStruct;

SQL_API_RC SQL_API_FN
  db2gDatabaseRestart (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef struct db2gRestartDbStruct
{
   db2Uint32 iDatabaseNameLen;
   db2Uint32 iUserIdLen;
   db2Uint32 iPasswordLen;
   db2Uint32 iTablespaceNamesLen;
   char *piDatabaseName;
   char *piUserId;
   char *piPassword;
   char *piTablespaceNames;
} db2gRestartDbStruct;
```

db2DatabaseRestart API parameters

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, pParamStruct.

pParamStruct
Input. A pointer to the db2RestartDbStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2RestartDbStruct data structure parameters

piDatabaseName
Input. A pointer to a string containing the alias of the database that is to be restarted.

piUserId
Input. A pointer to a string containing the user name of the application. May be NULL.

piPassword
Input. A pointer to a string containing a password for the specified user name (if any). May be NULL.

piTablespaceNames
Input. A pointer to a string containing a list of table space names to be dropped during the restart operation. May be NULL.

iOption
Input. Valid values are:

DB2_DB_SUSPEND_NONE
Performs normal crash recovery.

DB2_DB_RESUME_WRITE
Required to perform crash recovery on a database that has I/O writes suspended.

db2gRestartDbStruct data structure specific parameters

iDatabaseNameLen
Input. Length in bytes of piDatabaseName parameter.

iUserIdLen
Input. Length in bytes of piUserId parameter.

iPasswordLen
Input. Length in bytes of piPassword parameter.

iTablespaceNamesLen
Input. Length in bytes of piTablespaceNames parameter.

Usage notes

Call this API if an attempt to connect to a database returns an error message, indicating that the database must be restarted. This action occurs only if the previous session with this database terminated abnormally (due to power failure, for example).

At the completion of this API, a shared connection to the database is maintained if the user has CONNECT privilege, and an SQL warning is issued if any indoubt transactions exist. In this case, the database is still usable, but if the indoubt transactions are not resolved before the last connection to the database is dropped, another call to the API must be completed before the database can be used again.

In the case of circular logging, a database restart operation will fail if there is any problem with the table spaces, such as an I/O error, an unmounted file system, and so on. If losing such table spaces is not an issue, their names can be explicitly specified; this will put them into drop pending state, and the restart operation can complete successfully.

REXX API syntax

```
RESTART DATABASE database_alias [USER username USING password]
```

REXX API parameters

database_alias
Alias of the database to be restarted.

username
User name under which the database is to be restarted.

password
Password used to authenticate the user name.

db2DatabaseQuiesce - Quiesce the database

Forces all users off the database, immediately rolls back all active transactions or waits for them to complete their current units of work within the number of minutes specified (if they cannot be completed within the specified number of minutes, the operation will fail), and puts the database into quiesce mode. This API provides exclusive access to the database. During this quiesced period, system administration can be performed on the database by users with appropriate authority. After administration is complete, you can unquiesce the database, using the db2DatabaseUnquiesce API. The db2DatabaseUnquiesce API allows other users to connect to the database, without having to shut down and perform another database start. In this mode only groups or users with QUIESCE CONNECT authority and sysadm, sysmaint, or sysctrl will have access to the database and its objects.

Authorization

One of the following:

- sysadm
- dbadm

Required connection

Database

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2DatabaseQuiesce (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbQuiesceStruct
{
  char *piDatabaseName;
  db2Uint32 iImmediate;
  db2Uint32 iForce;
  db2Uint32 iTimeout;
} db2DbQuiesceStruct;

SQL_API_RC SQL_API_FN
  db2gDatabaseQuiesce (
```

```
            db2Uint32 versionNumber,
            void * pParmStruct,
            struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gDbQuiesceStruct
{
  db2Uint32 iDatabaseNameLen;
  char *piDatabaseName;
  db2Uint32 iImmediate;
  db2Uint32 iForce;
  db2Uint32 iTimeout;
} db2gDbQuiesceStruct;
```

db2DatabaseQuiesce API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2DbQuiesceStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2DbQuiesceStruct data structure parameters

piDatabaseName
Input. The database name.

iImmediate
Input. Valid values are:

TRUE=1
Force the applications immediately.

FALSE=0
Deferred force. Applications will wait the number of minutes specified by iTimeout parameter to let their current units of work be completed, and then will terminate. If this deferred force cannot be completed within the number of minutes specified by iTimeout parameter, the quiesce operation will fail.

iForce Input. Reserved for future use.

iTimeout
Input. Specifies the time, in minutes, to wait for applications to commit the current unit of work. If iTimeout is not specified, in a single-partition database environment, the default value is 10 minutes. In a partitioned database environment the value specified by the start_stop_time database manager configuration parameter will be used.

db2gDbQuiesceStruct data structure specific parameters

iDatabaseNameLen
Input. Specifies the length in bytes of piDatabaseName.

db2DatabaseUnquiesce - Unquiesce database

Restores user access to databases which have been quiesced for maintenance or other reasons. User access is restored without necessitating a shutdown and database restart.

Authorization

One of the following:

- sysadm
- dbadm

Required connection

Database

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2DatabaseUnquiesce (
              db2Uint32 versionNumber,
              void * pParmStruct,
              struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbUnquiesceStruct
{
              char *piDatabaseName;
} db2DbUnquiesceStruct;

SQL_API_RC SQL_API_FN
  db2gDatabaseUnquiesce (
              db2Uint32 versionNumber,
              void * pParmStruct,
              struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gDbUnquiesceStruct
{
              db2Uint32 iDatabaseNameLen;
              char *piDatabaseName;
} db2gDbUnquiesceStruct;
```

db2DatabaseUnquiesce API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2DbUnquiesceStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2DbUnquiesceStruct data structure parameters

piDatabaseName
Input. The database name.

db2gDbUnquiesceStruct data structure specific parameters

iDatabaseNameLen
Input. Specifies the length in bytes of piDatabaseName.

db2Export - Export data from a database

Exports data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables.

Authorization

One of the following:

- *dataaccess* authority
- CONTROL or SELECT privilege on each participating table or view

Label-based access control (LBAC) is enforced for this function. The data that is exported may be limited by the LBAC credentials of the caller if the data is protected by LBAC.

Required connection

Database. If implicit connect is enabled, a connection to the default database is established.

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Export (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ExportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqlu_media_list *piLobFileList;
   struct sqldcol *piDataDescriptor;
   struct sqllob *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2ExportOut *poExportInfoOut;
   struct db2ExportIn *piExportInfoIn;
   struct sqlu_media_list *piXmlPathList;
   struct sqlu_media_list *piXmlFileList;
} db2ExportStruct;

typedef SQL_STRUCTURE db2ExportIn
{
   db2Uint16 *piXmlSaveSchema;
} db2ExportIn;

typedef SQL_STRUCTURE db2ExportOut
{
   db2Uint64 oRowsExported;
} db2ExportOut;

SQL_API_RC SQL_API_FN
  db2gExport (
   db2Uint32 versionNumber,
```

```
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gExportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqlu_media_list *piLobFileList;
   struct sqldcol *piDataDescriptor;
   struct sqllob *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2ExportOut *poExportInfoOut;
   db2Uint16 iDataFileNameLen;
   db2Uint16 iFileTypeLen;
   db2Uint16 iMsgFileNameLen;
   struct db2ExportIn *piExportInfoIn;
   struct sqlu_media_list *piXmlPathList;
   struct sqlu_media_list *piXmlFileList;
} db2gExportStruct;
```

db2Export API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2ExportStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ExportStruct data structure parameters

piDataFileName
Input. A string containing the path and the name of the external file into which the data is to be exported.

piLobPathList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the LOB files are to be stored. Exported LOB data will be distributed evenly among all the paths listed in the sqlu_media_entry structure.

piLobFileList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_CLIENT_LOCATION, and its sqlu_location_entry structure containing base file names.

When the name space is exhausted using the first name in this list, the API will use the second name, and so on. When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from piLobPathList), and then appending a 3-digit sequence number and the .lob extension. For example, if the current LOB path is the directory /u/foo/lob/path, the current LOB file name is bar, and the LOBSINSEPFILES file type modifier is set, then the created LOB files will be /u/foo/LOB/path/bar.001.lob, /u/foo/LOB/path/bar.002.lob, and so on. If the LOBSINSEPFILES file

type modifier is not set, then all the LOB documents will be concatenated and put into one file /u/foo/lob/path/bar.001.lob

piDataDescriptor
Input. Pointer to an sqldcol structure specifying the column names for the output file. The value of the dcolmeth field determines how the remainder of the information provided in this parameter is interpreted by the export utility. Valid values for this parameter (defined in sqlutil header file, located in the include directory) are:

SQL_METH_N
Names. Specify column names to be used in the output file.

SQL_METH_D
Default. Existing column names from the table are to be used in the output file. In this case, the number of columns and the column specification array are both ignored. The column names are derived from the output of the SELECT statement specified in piActionString.

piActionString
Input. Pointer to an sqllob structure containing a valid dynamic SQL SELECT statement. The structure contains a 4-byte long field, followed by the characters that make up the SELECT statement. The SELECT statement specifies the data to be extracted from the database and written to the external file.

The columns for the external file (from piDataDescriptor), and the database columns from the SELECT statement, are matched according to their respective list/structure positions. The first column of data selected from the database is placed in the first column of the external file, and its column name is taken from the first element of the external column array.

piFileType
Input. A string that indicates the format of the data within the external file. Supported external file formats (defined in sqlutil header file) are:

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_WSF
Worksheet formats (WSF) for exchange with Lotus Symphony and 1-2-3 programs. Support for this file format is deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed.

SQL_IXF
PC version of the Integration Exchange Format, the preferred method for exporting data from a table. Data exported to this file format can later be imported or loaded into the same table or into another database manager table.

piFileTypeMod
Input. A pointer to an sqldcol structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types. See related link below: ″File type modifiers for the export utility.″

piMsgFileName
Input. A string containing the destination for error, warning, and informational messages returned by the utility. It can be the path and the name of an operating system file or a standard device. If the file already exists, the information is appended . If it does not exist, a file is created.

iCallerAction
Input. An action requested by the caller. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_INITIAL
Initial call. This value must be used on the first call to the API. If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested export operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility was not performed, and the utility is to terminate processing the initial request.

poExportInfoOut
A pointer to the db2ExportOut structure.

piExportInfoIn
Input. Pointer to the db2ExportIn structure.

piXmlPathList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the XML files are to be stored. Exported XML data will be distributed evenly among all the paths listed in the sqlu_media_entry structure.

piXmlFileList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_CLIENT_LOCATION, and its sqlu_location_entry structure containing base file names.

When the name space is exhausted using the first name in this list, the API will use the second name, and so on. When creating XML files during an export operation, file names are constructed by appending the current base name from this list to the current path (from piXmlFileList), and then appending a 3-digit sequence number and the .xml extension. For example, if the current XML path is the directory /u/foo/xml/path, the current XML file name is bar, and the XMLINSEPFILES file type modifier is set,

then the created XML files will be /u/foo/xml/path/bar.001.xml, /u/foo/xml/path/bar.002.xml, and so on. If the XMLINSEPFILES file type modifier is not set, then all the XML documents will be concatenated and put into one file /u/foo/xml/path/bar.001.xml

db2ExportIn data structure parameters

piXmlSaveSchema
Input. Indicates that the SQL identifier of the XML schema used to validate each exported XML document should be saved in the exported data file. Possible values are TRUE and FALSE.

db2ExportOut data structure parameters

oRowsExported
Output. Returns the number of records exported to the target file.

db2gExportStruct data structure specific parameters

iDataFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the data file name.

iFileTypeLen
Input. A 2-byte unsigned integer representing the length in bytes of the file type.

iMsgFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the message file name.

Usage notes

Before starting an export operation, you must complete all table operations and release all locks in one of two ways:

- Close all open cursors that were defined with the WITH HOLD clause, and commit the data changes by executing the COMMIT statement.
- Roll back the data changes by executing the ROLLBACK statement.

Table aliases can be used in the SELECT statement.

The messages placed in the message file include the information returned from the message retrieval service. Each message begins on a new line.

If the export utility produces warnings, the message will be written out to a message file, or standard output if one is not specified.

A warning message is issued if the number of columns (dcolnum field of sqldcol structure) in the external column name array, piDataDescriptor, is not equal to the number of columns generated by the SELECT statement. In this case, the number of columns written to the external file is the lesser of the two numbers. Excess database columns or external column names are not used to generate the output file.

If the db2uexpm.bnd module or any other shipped .bnd files are bound manually, the format option on the binder must not be used.

DB2 Connect can be used to export tables from DRDA servers such as DB2 for z/OS and OS/390, DB2 for VM and VSE, and DB2 for System i. Only PC/IXF export is supported.

PC/IXF import should be used to move data between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.

The export utility will not create multiple-part PC/IXF files when invoked from an AIX system.

Index definitions for a table are included in the PC/IXF file when the contents of a single database table are exported to a PC/IXF file with a piActionString parameter beginning with SELECT * FROM tablename, and the piDataDescriptor parameter specifying default names. Indexes are not saved for views, or if the SELECT clause of the piActionString includes a join. A WHERE clause, a GROUP BY clause, or a HAVING clause in the piActionString parameter will not prevent the saving of indexes. In all of these cases, when exporting from typed tables, the entire hierarchy must be exported.

The export utility will store the NOT NULL WITH DEFAULT attribute of the table in an IXF file if the SELECT statement provided is in the form: SELECT * FROM tablename.

When exporting typed tables, subselect statements can only be expressed by specifying the target table name and the WHERE clause. Fullselect and select-statement cannot be specified when exporting a hierarchy.

For file formats other than IXF, it is recommended that the traversal order list be specified, because it tells DB2 how to traverse the hierarchy, and what sub-tables to export. If this list is not specified, all tables in the hierarchy are exported, and the default order is the OUTER order. The alternative is to use the default order, which is the order given by the OUTER function.

Note: Use the same traverse order during an import operation. The load utility does not support loading hierarchies or sub-hierarchies.

REXX API syntax

```
EXPORT :stmt TO datafile OF filetype
[MODIFIED BY :filetmod] [USING :dcoldata]
MESSAGES msgfile [ROWS EXPORTED :number]


CONTINUE EXPORT


STOP EXPORT
```

REXX API parameters

stmt A REXX host variable containing a valid dynamic SQL SELECT statement. The statement specifies the data to be extracted from the database.

datafile
Name of the file into which the data is to be exported.

filetype
The format of the data in the export file. The supported file formats are:

DEL Delimited ASCII.

WSF Worksheet format. Support for this file format is deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed.

IXF PC version of Integration Exchange Format.

filetmod
A host variable containing additional processing options.

dcoldata
A compound REXX host variable containing the column names to be used in the export file. In the following, XXX represents the name of the host variable:

XXX.0 Number of columns (number of elements in the remainder of the variable).

XXX.1 First column name.

XXX.2 Second column name.

XXX.3 and so on.

If this parameter is NULL, or a value for dcoldata has not been specified, the utility uses the column names from the database table.

msgfile
File, path, or device name where error and warning messages are to be sent.

number
A host variable that will contain the number of exported rows.

db2Import - Import data into a table, hierarchy, nickname or view

Inserts data from an external file with a supported file format into a table, hierarchy, nickname or view. The load utility is faster than this function. The load utility, however, does not support loading data at the hierarchy level or loading into a nickname.

Authorization

- IMPORT using the INSERT option requires one of the following:
  - dataaccess
  - CONTROL privilege on each participating table, view or nickname
  - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the INSERT_UPDATE option, requires one of the following:
  - dataaccess
  - CONTROL privilege on the table, view or nickname
  - INSERT, SELECT, UPDATE and DELETE privilege on each participating table or view
- IMPORT to an existing table using the REPLACE or REPLACE_CREATE option, requires one of the following:
  - dataaccess

  - CONTROL privilege on the table or view
  - INSERT, SELECT, and DELETE privilege on the table or view
- IMPORT to a new table using the CREATE or REPLACE_CREATE option, requires one of the following:
  - dbadm
  - 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
- IMPORT to a table or a hierarchy that does not exist using the CREATE, or the REPLACE_CREATE option, requires one of the following:
  - dbadm
  - CREATETAB authority on the database, and one of:
    - IMPLICIT_SCHEMA authority on the database, if the schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema of the table exists
    - CONTROL privilege on every sub-table in the hierarchy, if the REPLACE_CREATE option on the entire hierarchy is used
- IMPORT to an existing hierarchy using the REPLACE option requires one of the following:
  - dataaccess
  - CONTROL privilege on every sub-table in the hierarchy

Required connection

Database. If implicit connect is enabled, a connection to the default database is established.

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Import (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ImportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2ImportIn *piImportInfoIn;
   struct db2ImportOut *poImportInfoOut;
   db2int32 *piNullIndicators;
   struct sqllob *piLongActionString;
} db2ImportStruct;
```

```
typedef SQL_STRUCTURE db2ImportIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   db2Uint64 iSkipcount;
   db2int32 *piCommitcount;
   db2Uint32 iWarningcount;
   db2Uint16 iNoTimeout;
   db2Uint16 iAccessLevel;
   db2Uint16 *piXmlParse;
   struct db2DMUXmlValidate *piXmlValidate;
} db2ImportIn;

typedef SQL_STRUCTURE db2ImportOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsInserted;
   db2Uint64 oRowsUpdated;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsCommitted;
} db2ImportOut;

typedef SQL_STRUCTURE db2DMUXmlMapSchema
{
   struct db2Char                       iMapFromSchema;
   struct db2Char                       iMapToSchema;
} db2DMUXmlMapSchema;

typedef SQL_STRUCTURE db2DMUXmlValidateXds
{
   struct db2Char *piDefaultSchema;
   db2Uint32 iNumIgnoreSchemas;
   struct db2Char *piIgnoreSchemas;
   db2Uint32 iNumMapSchemas;
   struct db2DMUXmlMapSchema *piMapSchemas;
} db2DMUXmlValidateXds;

typedef SQL_STRUCTURE db2DMUXmlValidateSchema
{
   struct db2Char *piSchema;
} db2DMUXmlValidateSchema;

typedef SQL_STRUCTURE db2DMUXmlValidate
{
   db2Uint16 iUsing;
   struct db2DMUXmlValidateXds *piXdsArgs;
   struct db2DMUXmlValidateSchema *piSchemaArgs;
} db2DMUXmlValidate;

SQL_API_RC SQL_API_FN
  db2gImport (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gImportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
```

```
   struct db2gImportIn *piImportInfoIn;
   struct dbg2ImportOut *poImportInfoOut;
   db2int32 *piNullIndicators;
   db2Uint16 iDataFileNameLen;
   db2Uint16 iFileTypeLen;
   db2Uint16 iMsgFileNameLen;
   struct sqllob *piLongActionString;
} db2gImportStruct;

typedef SQL_STRUCTURE db2gImportIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   db2Uint64 iSkipcount;
   db2int32 *piCommitcount;
   db2Uint32 iWarningcount;
   db2Uint16 iNoTimeout;
   db2Uint16 iAccessLevel;
   db2Uint16 *piXmlParse;
   struct db2DMUXmlValidate *piXmlValidate;
} db2gImportIn;

typedef SQL_STRUCTURE db2gImportOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsInserted;
   db2Uint64 oRowsUpdated;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsCommitted;
} db2gImportOut;
```

db2Import API parameters

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter pParmStruct.

pParmStruct
Input/Output. A pointer to the db2ImportStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ImportStruct data structure parameters

piDataFileName
Input. A string containing the path and the name of the external input file from which the data is to be imported.

piLobPathList
Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the LOB files can be found. This parameter is not valid when you import to a nickname.

piDataDescriptor
Input. Pointer to an sqldcol structure containing information about the columns being selected for import from the external file. The value of the dcolmeth field determines how the remainder of the information provided in this parameter is interpreted by the import utility. Valid values for this parameter are:

SQL_METH_N
Names. Selection of columns from the external input file is by column name.

SQL_METH_P
Positions. Selection of columns from the external input file is by column position.

SQL_METH_L
Locations. Selection of columns from the external input file is by column location. The database manager rejects an import call with a location pair that is invalid because of any one of the following conditions:

- Either the beginning or the ending location is not in the range from 1 to the largest signed 2-byte integer.
- The ending location is smaller than the beginning location.
- The input column width defined by the location pair is not compatible with the type and the length of the target column.

A location pair with both locations equal to zero indicates that a nullable column is to be filled with NULLs.

SQL_METH_D
Default. If piDataDescriptor is NULL, or is set to SQL_METH_D, default selection of columns from the external input file is done. In this case, the number of columns and the column specification array are both ignored. For DEL, IXF, or WSF files, the first n columns of data in the external input file are taken in their natural order, where n is the number of database columns into which the data is to be imported.

piActionString
Deprecated. Replaced by piLongActionString.

piLongActionString
Input. Pointer to an *sqllob* structure containing a 4-byte long field, followed by an array of characters specifying an action that affects the table.

The character array is of the form:

```
{INSERT | INSERT_UPDATE | REPLACE | CREATE | REPLACE_CREATE}
INTO {tname[(tcolumn-list)] |
[{ALL TABLES | (tname[(tcolumn-list)][, tname[(tcolumn-list)]])}]
[IN] HIERARCHY {STARTING tname | (tname[, tname])}
[UNDER sub-table-name | AS ROOT TABLE]}
```

INSERT
Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE
Adds the imported rows if their primary key values are not in the table, and uses them for update if their primary key values are found. This option is only valid if the target table has a primary key, and the specified (or implied) list of target columns being imported includes all columns for the primary key. This option cannot be applied to views.

REPLACE
Deletes all existing data from the table by truncating the table object, and inserts the imported data. The table definition and the index definitions are not changed. (Indexes are deleted and

replaced if indexixf is in FileTypeMod, and FileType is SQL_IXF.) If the table is not already defined, an error is returned.

Note: If an error occurs after the existing data is deleted, that data is lost.
This parameter is not valid when you import to a nickname.

CREATE

Note: The CREATE parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options CREATE and REPLACE_CREATE are deprecated".

Creates the table definition and the row contents using the information in the specified PC/IXF file, if the specified table is not defined. If the file was previously exported by DB2, indexes are also created. If the specified table is already defined, an error is returned. This option is valid for the PC/IXF file format only. This parameter is not valid when you import to a nickname.

REPLACE_CREATE

Note: The REPLACE_CREATE parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options CREATE and REPLACE_CREATE are deprecated".

Replaces the table contents using the PC/IXF row information in the PC/IXF file, if the specified table is defined. If the table is not already defined, the table definition and row contents are created using the information in the specified PC/IXF file. If the PC/IXF file was previously exported by DB2, indexes are also created. This option is valid for the PC/IXF file format only.

Note: If an error occurs after the existing data is deleted, that data is lost.
This parameter is not valid when you import to a nickname.

tname The name of the table, typed table, view, or object view into which the data is to be inserted. An alias for REPLACE, INSERT_UPDATE, or INSERT can be specified, except in the case of a server with a previous version of the DB2 product installed, when a qualified or unqualified name should be specified. If it is a view, it cannot be a read-only view.

tcolumn-list
A list of table or view column names into which the data is to be inserted. The column names must be separated by commas. If column names are not specified, column names as defined in the CREATE TABLE or the ALTER TABLE statement are used. If no column list is specified for typed tables, data is inserted into all columns within each sub-table.

sub-table-name
Specifies a parent table when creating one or more sub-tables under the CREATE option.

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the traversal-order-list.

HIERARCHY
Specifies that hierarchical data is to be imported.

STARTING
Keyword for hierarchy only. Specifies that the default order, starting from a given sub-table name, is to be used.

UNDER
Keyword for hierarchy and CREATE only. Specifies that the new hierarchy, sub-hierarchy, or sub-table is to be created under a given sub-table.

AS ROOT TABLE
Keyword for hierarchy and CREATE only. Specifies that the new hierarchy, sub-hierarchy, or sub-table is to be created as a stand-alone hierarchy.

The tname and the tcolumn-list parameters correspond to the tablename and the colname lists of SQL INSERT statements, and have the same restrictions.

The columns in tcolumn-list and the external columns (either specified or implied) are matched according to their position in the list or the structure (data from the first column specified in the sqldcol structure is inserted into the table or view field corresponding to the first element of the tcolumn-list).

If unequal numbers of columns are specified, the number of columns actually processed is the lesser of the two numbers. This could result in an error (because there are no values to place in some non-nullable table fields) or an informational message (because some external file columns are ignored).

This parameter is not valid when you import to a nickname.

piFileType
Input. A string that indicates the format of the data within the external file. Supported external file formats are:

SQL_ASC
Non-delimited ASCII.

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_IXF
PC version of the Integration Exchange Format, the preferred method for exporting data from a table so that it can be imported later into the same table or into another database manager table.

SQL_WSF
Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs. The WSF file type is not supported when you import to a nickname.

piFileTypeMod
Input. A pointer to a structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types. See related link "File type modifiers for the import utility".

piMsgFileName
Input. A string containing the destination for error, warning, and informational messages returned by the utility. It can be the path and the name of an operating system file or a standard device. If the file already exists, it is appended to. If it does not exist, a file is created.

iCallerAction
Input. An action requested by the caller. Valid values are:

SQLU_INITIAL
Initial call. This value must be used on the first call to the API. If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested import operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility was not performed, and the utility is to terminate processing the initial request.

piImportInfoIn
Input. Pointer to the db2ImportIn structure.

poImportInfoOut
Output. Pointer to the db2ImportOut structure.

piNullIndicators
Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. The number of elements in this array must match the number of columns in the input file; there is a one-to-one ordered correspondence between the elements of this array and the columns being imported from the data file. Therefore, the number of elements must equal the dcolnum field of the piDataDescriptor parameter. Each element of the array contains a number identifying a column in the data file that is to be used as a null indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified column in the data file must contain a Y or an N. A Y indicates that the table column data is NULL, and N indicates that the table column data is not NULL.

piXmlPathList
Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the XML files can be found.

db2ImportIn data structure parameters

iRowcount
Input. The number of physical records to be loaded. Allows a user to load only the first iRowcount rows in a file. If iRowcount is 0, import will attempt to process all the rows from the file.

iRestartcount
Input. The number of records to skip before starting to insert or update records. Functionally equivalent to iSkipcount parameter. iRestartcount and iSkipcount parameters are mutually exclusive.

iSkipcount
Input. The number of records to skip before starting to insert or update records. Functionally equivalent to iRestartcount.

piCommitcount
Input. The number of records to import before committing them to the database. A commit is performed whenever piCommitcount records are imported. A NULL value specifies the default commit count value, which is zero for offline import and AUTOMATIC for online import. Commitcount AUTOMATIC is specified by passing in the value DB2IMPORT_COMMIT_AUTO.

iWarningcount
Input. Stops the import operation after iWarningcount warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the import file or the target table is specified incorrectly, the import utility will generate a warning for each row that it attempts to import, which will cause the import to fail.

If iWarningcount is 0, or this option is not specified, the import operation will continue regardless of the number of warnings issued.

iNoTimeout
Input. Specifies that the import utility will not time out while waiting for locks. This option supersedes the locktimeout database configuration parameter. Other applications are not affected. Valid values are:

DB2IMPORT_LOCKTIMEOUT
Indicates that the value of the locktimeout configuration parameter is respected.

DB2IMPORT_NO_LOCKTIMEOUT
Indicates there is no timeout.

iAccessLevel
Input. Specifies the access level. Valid values are:

- SQLU_ALLOW_NO_ACCESS
Specifies that the import utility locks the table exclusively.

- SQLU_ALLOW_WRITE_ACCESS
Specifies that the data in the table should still be accessible to readers and writers while the import is in progress.

An intent exclusive (IX) lock on the target table is acquired when the first row is inserted. This allows concurrent readers and writers to access table data. Online mode is not compatible with the REPLACE, CREATE, or REPLACE_CREATE import options. Online mode is not supported in conjunction with buffered inserts. The import operation will periodically commit inserted data to prevent lock escalation to a table lock and to avoid running out of active log space. These commits will be performed even if the piCommitCount parameter was not used. During each commit, import will lose its IX table lock, and will attempt to reacquire it after the commit. This parameter is required when you import to a nickname and piCommitCount parameter must be specified with a valid number (AUTOMATIC is not considered a valid option).

piXmlParse
Input. Type of parsing that should occur for XML documents. Valid values found in the db2ApiDf header file in the include directory, are:

DB2DMU_XMLPARSE_PRESERVE_WS
Whitespace should be preserved.

DB2DMU_XMLPARSE_STRIP_WS
Whitespace should be stripped.

piXmlValidate
Input. Pointer to the db2DMUXmlValidate structure. Indicates that XML schema validation should occur for XML documents.

db2ImportOut data structure parameters

oRowsRead
Output. Number of records read from the file during import.

oRowsSkipped
Output. Number of records skipped before inserting or updating begins.

oRowsInserted
Output. Number of rows inserted into the target table.

oRowsUpdated
Output. Number of rows in the target table updated with information from the imported records (records whose primary key value already exists in the table).

oRowsRejected
Output. Number of records that could not be imported.

oRowsCommitted
Output. Number of records imported successfully and committed to the database.

db2DMUXmlMapSchema data structure parameters

iMapFromSchema
Input. The SQL identifier of the XML schema to map from.

iMapToSchema
Input. The SQL identifier of the XML schema to map to.

db2DMUXmlValidateXds data structure parameters

piDefaultSchema
Input. The SQL identifier of the XML schema that should be used for validation when an XDS does not contain an SCH attribute.

iNumIgnoreSchemas
Input. The number of XML schemas that will be ignored during XML schema validation if they are referred to by an SCH attribute in XDS.

piIgnoreSchemas
Input. The list of XML schemas that will be ignored during XML schema validation if they are referred to by an SCH attribute in XDS.

iNumMapSchemas
Input. The number of XML schemas that will be mapped during XML schema validation. The first schema in the schema map pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

piMapSchemas
Input. The list of XML schema pairs, where each pair represents a mapping of one schema to a different one. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

db2DMUXmlValidateSchema data structure parameters

piSchema
Input. The SQL identifier of the XML schema to use.

db2DMUXmlValidate data structure parameters

iUsing
Input. A specification of what to use to perform XML schema validation. Valid values found in the db2ApiDf header file in the include directory, are:

- **DB2DMU_XMLVAL_XDS**
  Validation should occur according to the XDS. This corresponds to the CLP ″XMLVALIDATE USING XDS″ clause.
- **DB2DMU_XMLVAL_SCHEMA**
  Validation should occur according to a specified schema. This corresponds to the CLP ″XMLVALIDATE USING SCHEMA″ clause.
- **DB2DMU_XMLVAL_SCHEMALOC_HINTS**
  Validation should occur according to schemaLocation hints found within the XML document. This corresponds to the ″XMLVALIDATE USING SCHEMALOCATION HINTS″ clause.

piXdsArgs
Input. Pointer to a db2DMUXmlValidateXds structure, representing arguments that correspond to the CLP ″XMLVALIDATE USING XDS″ clause.

This parameter applies only when the iUsing parameter in the same structure is set to DB2DMU_XMLVAL_XDS.

piSchemaArgs
Input. Pointer to a db2DMUXmlValidateSchema structure, representing arguments that correspond to the CLP ″XMLVALIDATE USING SCHEMA″ clause.

This parameter applies only when the iUsing parameter in the same structure is set to DB2DMU_XMLVAL_SCHEMA.

db2gImportStruct data structure specific parameters

iDataFileNameLen
Input. Specifies the length in bytes of piDataFileName parameter.

iFileTypeLen
Input. Specifies the length in bytes of piFileType parameter.

iMsgFileNameLen
Input. Specifies the length in bytes of piMsgFileName parameter.

Usage notes

Before starting an import operation, you must complete all table operations and release all locks in one of two ways:

- Close all open cursors that were defined with the WITH HOLD clause, and commit the data changes by executing the COMMIT statement.
- Roll back the data changes by executing the ROLLBACK statement.

The import utility adds rows to the target table using the SQL INSERT statement.

The utility issues one INSERT statement for each row of data in the input file. If an INSERT statement fails, one of two actions result:

- If it is likely that subsequent INSERT statements can be successful, a warning message is written to the message file, and processing continues.
- If it is likely that subsequent INSERT statements will fail, and there is potential for database damage, an error message is written to the message file, and processing halts.

The utility performs an automatic COMMIT after the old rows are deleted during a REPLACE or a REPLACE_CREATE operation. Therefore, if the system fails, or the application interrupts the database manager after the table object is truncated, all of the old data is lost. Ensure that the old data is no longer needed before using these options.

If the log becomes full during a CREATE, REPLACE, or REPLACE_CREATE operation, the utility performs an automatic COMMIT on inserted records. If the system fails, or the application interrupts the database manager after an automatic COMMIT, a table with partial data remains in the database. Use the REPLACE or the REPLACE_CREATE option to rerun the whole import operation, or use INSERT with the iRestartcount parameter set to the number of rows successfully imported.

By default, automatic COMMITs are not performed for the INSERT or the INSERT_UPDATE option. They are, however, performed if the *piCommitcount parameter is not zero. A full log results in a ROLLBACK.

Whenever the import utility performs a COMMIT, two messages are written to the message file: one indicates the number of records to be committed, and the other is

written after a successful COMMIT. When restarting the import operation after a failure, specify the number of records to skip, as determined from the last successful COMMIT.

The import utility accepts input data with minor incompatibility problems (for example, character data can be imported using padding or truncation, and numeric data can be imported with a different numeric data type), but data with major incompatibility problems is not accepted.

One cannot REPLACE or REPLACE_CREATE an object table if it has any dependents other than itself, or an object view if its base table has any dependents (including itself). To replace such a table or a view, do the following:

1. Drop all foreign keys in which the table is a parent.
2. Run the import utility.
3. Alter the table to recreate the foreign keys.

If an error occurs while recreating the foreign keys, modify the data to maintain referential integrity.

Referential constraints and foreign key definitions are not preserved when creating tables from PC/IXF files. (Primary key definitions are preserved if the data was previously exported using SELECT *.)

Importing to a remote database requires enough disk space on the server for a copy of the input data file, the output message file, and potential growth in the size of the database.

If an import operation is run against a remote database, and the output message file is very long (more than 60 KB), the message file returned to the user on the client may be missing messages from the middle of the import operation. The first 30 KB of message information and the last 30 KB of message information are always retained.

Non-default values for piDataDescriptor, or specifying an explicit list of table columns in piLongActionString, makes importing to a remote database slower.

The database table or hierarchy must exist before data in the ASC, DEL, or WSF file formats can be imported; however, if the table does not already exist, IMPORT CREATE or IMPORT REPLACE_CREATE creates the table when it imports data from a PC/IXF file. For typed tables, IMPORT CREATE can create the type hierarchy and the table hierarchy as well.

PC/IXF import should be used to move data (including hierarchical data) between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.

The data in ASC and DEL files is assumed to be in the code page of the client application performing the import. PC/IXF files, which allow for different code pages, are recommended when importing data in different code pages. If the PC/IXF file and the import utility are in the same code page, processing occurs as for a regular application. If the two differ, and the FORCEIN option is specified, the import utility assumes that data in the PC/IXF file has the same code page as the application performing the import. This occurs even if there is a conversion table for the two code pages. If the two differ, the FORCEIN option is not

specified, and there is a conversion table, all data in the PC/IXF file will be converted from the file code page to the application code page. If the two differ, the FORCEIN option is not specified, and there is no conversion table, the import operation will fail. This applies only to PC/IXF files on DB2 for AIX clients.

For table objects on an 8KB page that are close to the limit of 1012 columns, import of PC/IXF data files may cause DB2 to return an error, because the maximum size of an SQL statement was exceeded. This situation can occur only if the columns are of type CHAR, VARCHAR, or CLOB. The restriction does not apply to import of DEL or ASC files.

DB2 Connect can be used to import data to DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF import (INSERT option) is supported. The restartcnt parameter, but not the commitcnt parameter, is also supported.

When using the CREATE option with typed tables, create every sub-table defined in the PC/IXF file; sub-table definitions cannot be altered. When using options other than CREATE with typed tables, the traversal order list enables one to specify the traverse order; therefore, the traversal order list must match the one used during the export operation. For the PC/IXF file format, one need only specify the target sub-table name, and use the traverse order stored in the file. The import utility can be used to recover a table previously exported to a PC/IXF file. The table returns to the state it was in when exported.

Data cannot be imported to a system table, a declared temporary table, a created temporary table, or a summary table.

Views cannot be created through the import utility.

On the Windows operating system:

- Importing logically split PC/IXF files is not supported.
- Importing bad format PC/IXF or WSF files is not supported.

Federated considerations

When using the db2Import API and the INSERT, UPDATE, or INSERT_UPDATE parameters, you must ensure that you have CONTROL privilege on the participating nickname. You must ensure that the nickname you wish to use when doing an import operation already exists.

db2Inspect - Inspect database for architectural integrity

Inspects the database for architectural integrity and checks the pages of the database for page consistency.

Scope

In a single partition database environment, the scope is the single database partition only. In a partitioned database environment it is the collection of all logical database partitions defined in `db2nodes.cfg`. For partitioned tables, the scope for database and table space level inspection includes individual data partitions and non-partitioned indexes. Table level inspection for a partitioned table checks all the data partitions and indexes in a table, rather than checking a single data partition or index.

Authorization

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- CONTROL privilege on the table

Required connection

Database

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Inspect (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InspectStruct
{
   char *piTablespaceName;
   char *piTableName;
   char *piSchemaName;
   char *piResultsName;
   char *piDataFileName;
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint32 iAction;
   db2int32 iTablespaceID;
   db2int32 iObjectID;
   db2Uint32 iFirstPage;
   db2Uint32 iNumberOfPages;
   db2Uint32 iFormatType;
   db2Uint32 iOptions;
   db2Uint32 iBeginCheckOption;
   db2int32 iLimitErrorReported;
   db2Uint16 iObjectErrorState;
   db2Uint16 iCatalogToTablespace;
   db2Uint16 iKeepResultfile;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   db2Uint16 iLevelObjectData;
   db2Uint16 iLevelObjectIndex;
   db2Uint16 iLevelObjectLong;
   db2Uint16 iLevelObjectLOB;
   db2Uint16 iLevelObjectBlkMap;
   db2Uint16 iLevelExtentMap;
   db2Uint16 iLevelObjectXML;
   db2Uint32 iLevelCrossObject;
} db2InspectStruct;

SQL_API_RC SQL_API_FN
  db2gInspect (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInspectStruct
```

```
{
   char *piTablespaceName;
   char *piTableName;
   char *piSchemaName;
   char *piResultsName;
   char *piDataFileName;
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint32 iResultsNameLength;
   db2Uint32 iDataFileNameLength;
   db2Uint32 iTablespaceNameLength;
   db2Uint32 iTableNameLength;
   db2Uint32 iSchemaNameLength;
   db2Uint32 iAction;
   db2int32 iTablespaceID;
   db2int32 iObjectID;
   db2Uint32 iFirstPage;
   db2Uint32 iNumberOfPages;
   db2Uint32 iFormatType;
   db2Uint32 iOptions;
   db2Uint32 iBeginCheckOption;
   db2int32 iLimitErrorReported;
   db2Uint16 iObjectErrorState;
   db2Uint16 iCatalogToTablespace;
   db2Uint16 iKeepResultfile;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   db2Uint16 iLevelObjectData;
   db2Uint16 iLevelObjectIndex;
   db2Uint16 iLevelObjectLong;
   db2Uint16 iLevelObjectLOB;
   db2Uint16 iLevelObjectBlkMap;
   db2Uint16 iLevelExtentMap;
   db2Uint16 iLevelObjectXML;
   db2Uint32 iLevelCrossObject;
} db2gInspectStruct;
```

db2Inspect API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2InspectStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2InspectStruct data structure parameters

piTablespaceName
Input. A string containing the table space name. The table space must be identified for operations on a table space. If the pointer is NULL, the table space ID value is used as input.

piTableName
Input. A string containing the table name. The table must be identified for operations on a table or a table object. If the pointer is NULL, the table space ID and table object ID values are used as input.

piSchemaName
Input. A string containing the schema name.

piResultsName
Input. A string containing the name for results output file. This input must be provided. The file will be written out to the diagnostic data directory path.

piDataFileName
Input. Reserved for future use. Must be set to NULL.

piNodeList
Input. A pointer to an array of database partition numbers on which to perform the operation.

iAction
Input. Specifies the inspect action. Valid values (defined in the db2ApiDf header file, which is located in the include directory) are:

DB2INSPECT_ACT_CHECK_DB
Inspect the entire database.

DB2INSPECT_ACT_CHECK_TABSPACE
Inspect a table space.

DB2INSPECT_ACT_CHECK_TABLE
Inspect a table.

DB2INSPECT_ACT_FORMAT_XML
Format an XML object page.

DB2INSPECT_ACT_ROWCMPEST_TBL
Estimate row compression effectiveness on a table.

iTablespaceID
Input. Specifies the table space ID. If the table space must be identified, the table space ID value is used as input if the pointer to table space name is NULL.

iObjectID
Input. Specifies the object ID. If the table must be identified, the object ID value is used as input if the pointer to table name is NULL.

iBeginCheckOption
Input. Option for check database or check table space operation to indicate where operation should begin. It must be set to zero to begin from the normal start. Values are:

DB2INSPECT_BEGIN_TSPID
Use this value for check database to begin with the table space specified by the table space ID field, the table space ID must be set.

DB2INSPECT_BEGIN_TSPID_OBJID
Use this value for check database to begin with the table specified by the table space ID and object ID field. To use this option, the table space ID and object ID must be set.

DB2INSPECT_BEGIN_OBJID
Use this value for check table space to begin with the table specified by the object ID field, the object ID must be set.

iLimitErrorReported
Input. Specifies the reporting limit of the number of pages in error for an object. Specify the number you want to use as the limit value or specify one the following values:

DB2INSPECT_LIMIT_ERROR_DEFAULT
Use this value to specify that the maximum number of pages in error to be reported is the extent size of the object.

DB2INSPECT_LIMIT_ERROR_ALL
Use this value to report all pages in error.

When DB2INSPECT_LVL_XOBJ_INXDAT_RID is used in the iLevelCrossObject field, the limit value specified, or the above DEFAULT or ALL values, represent a limit in the number of errors, instead of number of pages in error, to be reported during the online index to data consistency checking.

iObjectErrorState
Input. Specifies whether to scan objects in error state. Valid values are:

DB2INSPECT_ERROR_STATE_NORMAL
Process object only in normal state.

DB2INSPECT_ERROR_STATE_ALL
Process all objects, including objects in error state.

When DB2INSPECT_LVL_XOBJ_INXDAT_RID is used in the iLevelCrossObject field, as long as the index or data object is in an error state, DB2INSPECT_ERROR_STATE_ALL will be ignored if specified in this field, and the online index to data consistency checking will not be performed.

iKeepResultfile
Input. Specifies result file retention. Valid values are:

DB2INSPECT_RESFILE_CLEANUP
If errors are reported, the result output file will be retained. Otherwise, the result file will be removed at the end of the operation.

DB2INSPECT_RESFILE_KEEP_ALWAYS
The result output file will be retained.

iAllNodeFlag
Input. Indicates whether the operation is to be applied to all nodes defined in `db2nodes.cfg`. Valid values are:

DB2_NODE_LIST
Apply to all nodes in a node list that is passed in pNodeList.

DB2_ALL_NODES
Apply to all nodes. pNodeList should be NULL. This is the default value.

DB2_ALL_EXCEPT
Apply to all nodes except those in a node list that is passed in pNodeList.

iNumNodes
Input. Specifies the number of nodes in the pNodeList array.

iLevelObjectData
Input. Specifies processing level for data object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectIndex
Input. Specifies processing level for index object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectLong
Input. Specifies processing level for long object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectLOB
Input. Specifies processing level for LOB object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectBlkMap
Input. Specifies processing level for block map object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelExtentMap
Input. Specifies processing level for extent map. Valid values (defined in the db2ApiDf header file, which is located in the include directory) are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectXML
Input. Specifies processing level for XML object. Valid values (defined in the db2ApiDf header file, which is located in the include directory) are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelCrossObject
A bit-based field used for any cross object consistency checking. Valid values are:

DB2INSPECT_LVL_XOBJ_NONE
Online index data consistency checking will not be performed (0x00000000).

DB2INSPECT_LVL_XOBJ_INXDAT_RID
INDEXDATA checking is enabled on RID index (0x00000001) and will be performed with IS table lock to allow for both readers and writers.

db2gInspectStruct data structure specific parameters

iResultsNameLength
Input. The string length of the results file name.

iDataFileNameLength
Input. The string length of the data output file name.

iTablespaceNameLength
Input. The string length of the table space name.

iTableNameLength
Input. The string length of the table name.

iSchemaNameLength
Input. The string length of the schema name.

Usage notes

The online inspect processing will access database objects using isolation level uncommitted read. Commit processing will be done during the inspect processing. It is advisable to end the unit of work by committing or rolling back changes, by executing a COMMIT or ROLLBACK statement respectively, before starting the inspect operation.

The inspect check processing will write out unformatted inspection data results to the result file. The file will be written out to the diagnostic data directory path. If there are no errors found by the check processing, the result output file will be erased at the end of the inspect operation. If there are errors found by the check processing, the result output file will not be erased at the end of the inspect operation. To see the inspection details, format the inspection result output file with the db2inspf utility.

In a partitioned database environment, the extension of the result output file will correspond to the database partition number. The file is located in the database manager diagnostic data directory path.

A unique results output file name must be specified. If the result output file already exists, the operation will not be processed.

When you call the db2Inspect API, you need to specify `iLevelCrossObject` in the db2InspectStruct with a proper value. When DB2INSPECT_LVL_XOBJ_NONE is used, online index data consistency checking will not be performed. To enable online index data consistency checking, DB2INSPECT_LVL_XOBJ_INXDAT_RID needs to be specified in the `iLevelCrossObject` field.

The processing of table spaces will process only the objects that reside in that table space. The exception is during an index data consistency check, when data objects can reside in other table spaces and still benefit from the checking, as long as the index objects are in the table space to be inspected. For a partitioned table, each index can reside in a different table space. Only those indexes that reside in the specified table space will benefit from the index to data checking.

db2InstanceQuiesce - Quiesce instance

Forces all users off the instance, immediately rolls back all active transaction, and puts the database into quiesce mode. This API provides exclusive access to the instance. During this quiesced period, system administration can be performed on the instance. After administration is complete, you can unquiesce the database using the db2DatabaseUnquiesce API. This API allows other users to connect to the database without having to shut down and perform another database start.

In this mode only groups or users with QUIESCE CONNECT authority and sysadm, sysmaint, or sysctrl authority will have access to the database and its objects.

Authorization

One of the following:

- sysadm
- sysctrl

Required connection

None

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2InstanceQuiesce (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InsQuiesceStruct
{
             char *piInstanceName;
```

```
            char *piUserId;
            char *piGroupId;
            db2Uint32 iImmediate;
            db2Uint32 iForce;
            db2Uint32 iTimeout;
} db2InsQuiesceStruct;

SQL_API_RC SQL_API_FN
  db2gInstanceQuiesce (
            db2Uint32 versionNumber,
            void * pParmStruct,
            struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInsQuiesceStruct
{
            db2Uint32 iInstanceNameLen;
            char *piInstanceName;
            db2Uint32 iUserIdLen;
            char *piUserId;
            db2Uint32 iGroupIdLen;
            char *piGroupId;
            db2Uint32 iImmediate;
            db2Uint32 iForce;
            db2Uint32 iTimeout;
} db2gInsQuiesceStruct;
```

db2InstanceQuiesce API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2InsQuiesceStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2InsQuiesceStruct data structure parameters

piInstanceName
Input. The instance name.

piUserId
Input. The name of the a user who will be allowed access to the instance while it is quiesced.

piGroupId
Input. The name of a group that will be allowed access to the instance while the instance is quiesced.

iImmediate
Input. Valid values are:

TRUE=1
Force the applications immediately.

FALSE=0
Deferred force. Applications will wait the number of minutes specified by iTimeout parameter to let their current units of work be completed, and then will terminate. If this deferred force cannot be completed within the number of minutes specified by iTimeout parameter, the quiesce operation will fail.

iForce Input. Reserved for future use.

iTimeout
Input. Specifies the time, in minutes, to wait for applications to commit the current unit of work. If iTimeout is not specified, in a single-partition database environment, the default value is 10 minutes. In a partitioned database environment the value specified by the start_stop_time database manager configuration parameter will be used.

db2gInsQuiesceStruct data structure specific parameters

iInstanceNameLen
Input. Specifies the length in bytes of piInstanceName.

iUserIdLen
Input. Specifies the length in bytes of piUserID.

iGroupIdLen
Input. Specifies the length in bytes of piGroupId.

db2InstanceStart - Start instance

Starts a local or remote instance.

Scope

In a single-partition database environment, the scope is that single database partition only. In a partitioned database environment, it is the collection of all logical database partition servers defined in the node configuration file, `db2nodes.cfg`.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection

None

API include file

`db2ApiDf.h`

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2InstanceStart (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InstanceStartStruct
{
             db2int8 iIsRemote;
             char *piRemoteInstName;
             db2DasCommData * piCommData;
             db2StartOptionsStruct * piStartOpts;
} db2InstanceStartStruct;
```

```
typedef SQL_STRUCTURE db2DasCommData
{
             db2int8 iCommParam;
             char *piNodeOrHostName;
             char *piUserId;
             char *piUserPw;
} db2DasCommData;

typedef SQL_STRUCTURE db2StartOptionsStruct
{
             db2Uint32 iIsProfile;
             char *piProfile;
             db2Uint32 iIsNodeNum;
             db2NodeType iNodeNum;
             db2Uint32 iOption;
             db2Uint32 iIsHostName;
             char *piHostName;
             db2Uint32 iIsPort;
             db2PortType iPort;
             db2Uint32 iIsNetName;
             char *piNetName;
             db2Uint32 iTblspaceType;
             db2NodeType iTblspaceNode;
             db2Uint32 iIsComputer;
             char *piComputer;
             char *piUserName;
             char *piPassword;
             db2QuiesceStartStruct iQuiesceOpts;
} db2StartOptionsStruct;

typedef SQL_STRUCTURE db2QuiesceStartStruct
{
             db2int8 iIsQRequested;
             char *piQUsrName;
             char *piQGrpName;
             db2int8 iIsQUsrGrpDef;
} db2QuiesceStartStruct;

SQL_API_RC SQL_API_FN
  db2gInstanceStart (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInstanceStStruct
{
             db2int8 iIsRemote;
             db2Uint32 iRemoteInstLen;
             char *piRemoteInstName;
             db2gDasCommData * piCommData;
             db2gStartOptionsStruct * piStartOpts;
} db2gInstanceStStruct;

typedef SQL_STRUCTURE db2gDasCommData
{
             db2int8 iCommParam;
             db2Uint32 iNodeOrHostNameLen;
             char *piNodeOrHostName;
             db2Uint32 iUserIdLen;
             char *piUserId;
             db2Uint32 iUserPwLen;
             char *piUserPw;
} db2gDasCommData;

typedef SQL_STRUCTURE db2gStartOptionsStruct
{
             db2Uint32 iIsProfile;
```

```
            char *piProfile;
            db2Uint32 iIsNodeNum;
            db2NodeType iNodeNum;
            db2Uint32 iOption;
            db2Uint32 iIsHostName;
            char *piHostName;
            db2Uint32 iIsPort;
            db2PortType iPort;
            db2Uint32 iIsNetName;
            char *piNetName;
            db2Uint32 iTblspaceType;
            db2NodeType iTblspaceNode;
            db2Uint32 iIsComputer;
            char *piComputer;
            char *piUserName;
            char *piPassword;
            db2gQuiesceStartStruct iQuiesceOpts;
} db2gStartOptionsStruct;

typedef SQL_STRUCTURE db2gQuiesceStartStruct
{
            db2int8 iIsQRequested;
            db2Uint32 iQUsrNameLen;
            char *piQUsrName;
            db2Uint32 iQGrpNameLen;
            char *piQGrpName;
            db2int8 iIsQUsrGrpDef;
} db2gQuiesceStartStruct;
```

db2InstanceStart API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2InstanceStartStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2InstanceStartStruct data structure parameters

iIsRemote
Input. An indicator set to constant integer value TRUE or FALSE. This parameter should be set to TRUE if this is a remote start.

piRemoteInstName
Input. The name of the remote instance.

piCommData
Input. A pointer to the db2DasCommData structure.

piStartOpts
Input. A pointer to the db2StartOptionsStruct structure.

db2DasCommData data structure parameters

iCommParam
Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if this is a remote start.

piNodeOrHostName
Input. The database partition or hostname.

piUserId
Input. The user name.

piUserPw
Input. The user password.

db2StartOptionsStruct data structure parameters

iIsProfile
Input. Indicates whether a profile is specified. If this field indicates that a profile is not specified, the file db2profile is used.

piProfile
Input. The name of the profile file to be executed at each node to define the DB2 environment (MPP only). This file is executed before the nodes are started. The default value is db2profile.

iIsNodeNum
Input. Indicates whether a node number is specified. If specified, the start command only affects the specified node.

iNodeNum
Input. The database partition number.

iOption
Input. Specifies an action. Valid values for OPTION (defined in sqlenv header file, located in the include directory) are:

SQLE_NONE
Issue the normal db2start operation.

SQLE_ADDNODE
Issue the ADD NODE command.

SQLE_RESTART
Issue the RESTART DATABASE command.

SQLE_RESTART_PARALLEL
Issue the RESTART DATABASE command for parallel execution.

SQLE_STANDALONE
Start the node in STANDALONE mode.

iIsHostName
Input. Indicates whether a host name is specified.

piHostName
Input. The system name.

iIsPort
Input. Indicates whether a port number is specified.

iPort Input. The port number.

iIsNetName
Input. Indicates whether a net name is specified.

piNetName
Input. The network name.

iTblspaceType
Input. Specifies the type of system temporary table space definitions to be used for the node being added. Valid values are:

SQLE_TABLESPACES_NONE
Do not create any system temporary table spaces.

SQLE_TABLESPACES_LIKE_NODE
The containers for the system temporary table spaces should be the same as those for the specified node.

SQLE_TABLESPACES_LIKE_CATALOG
The containers for the system temporary table spaces should be the same as those for the catalog node of each database.

iTblspaceNode
Input. Specifies the node number from which the system temporary table space definitions should be obtained. The node number must exist in the db2nodes.cfg file, and is only used if the tblspace_type field is set to SQLE_TABLESPACES_LIKE_NODE.

iIsComputer
Input. Indicates whether a computer name is specified. Valid on the Windows operating system only.

piComputer
Input. Computer name. Valid on the Windows operating system only.

piUserName
Input. Logon account user name. Valid on the Windows operating system only.

piPassword
Input. The password corresponding to the logon account user name.

iQuiesceOpts
Input. A pointer to the db2QuiesceStartStruct structure.

db2QuiesceStartStruct data structure parameters

iIsQRequested
Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if quiesce is requested.

piQUsrName
Input. The quiesced username.

piQGrpName
Input. The quiesced group name.

iIsQUsrGrpDef
Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if a quiesced user or quiesced group is defined.

db2gInstanceStStruct data structure specific parameters

iRemoteInstLen
Input. Specifies the length in bytes of piRemoteInstName.

db2gDasCommData data structure specific parameters

iNodeOrHostNameLen
Input. Specifies the length in bytes of piNodeOrHostName.

iUserIdLen
Input. Specifies the length in bytes of piUserId.

iUserPwLen
Input. Specifies the length in bytes of piUserPw.

db2gQuiesceStartStruct data structure specific parameters

iQUsrNameLen
Input. Specifies the length in bytes of piQusrName.

iQGrpNameLen
Input. Specifies the length in bytes of piQGrpName.

db2InstanceStop - Stop instance

Stops the local or remote DB2 instance.

Scope

In a single-partition database environment, the scope is that single database partition only. In a partitioned database environment, it is the collection of all logical database partition servers defined in the node configuration file, db2nodes.cfg.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection

None

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2InstanceStop (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InstanceStopStruct
{
             db2int8 iIsRemote;
             char *piRemoteInstName;
             db2DasCommData * piCommData;
             db2StopOptionsStruct * piStopOpts;
} db2InstanceStopStruct;

typedef SQL_STRUCTURE db2DasCommData
{
             db2int8 iCommParam;
             char *piNodeOrHostName;
             char *piUserId;
             char *piUserPw;
} db2DasCommData;
```

```
typedef SQL_STRUCTURE db2StopOptionsStruct
{
             db2Uint32 iIsProfile;
             char *piProfile;
             db2Uint32 iIsNodeNum;
             db2NodeType iNodeNum;
             db2Uint32 iStopOption;
             db2Uint32 iCallerac;
} db2StopOptionsStruct;

SQL_API_RC SQL_API_FN
  db2gInstanceStop (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInstanceStopStruct
{
             db2int8 iIsRemote;
             db2Uint32 iRemoteInstLen;
             char *piRemoteInstName;
             db2gDasCommData * piCommData;
             db2StopOptionsStruct * piStopOpts;
} db2gInstanceStopStruct;

typedef SQL_STRUCTURE db2gDasCommData
{
             db2int8 iCommParam;
             db2Uint32 iNodeOrHostNameLen;
             char *piNodeOrHostName;
             db2Uint32 iUserIdLen;
             char *piUserId;
             db2Uint32 iUserPwLen;
             char *piUserPw;
} db2gDasCommData;
```

db2InstanceStop API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2InstanceStopStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2InstanceStopStruct data structure parameters

iIsRemote
Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if this is a remote start.

piRemoteInstName
Input. The name of the remote instance.

piCommData
Input. A pointer to the db2DasCommData structure.

piStopOpts
Input. A pointer to the db2StopOptionsStruct structure.

db2DasCommData data structure parameters

iCommParam
Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if this is a remote start.

piNodeOrHostName
Input. The database partition or hostname.

piUserId
Input. The user name.

piUserPw
Input. The user password.

db2StopOptionsStruct data structure parameters

iIsProfile
Input. Indicates whether a profile is specified. Possible values are TRUE and FALSE. If this field indicates that a profile is not specified, the file db2profile is used.

piProfile
Input. The name of the profile file that was executed at startup to define the DB2 environment for those nodes that were started (MPP only). If a profile for the db2InstanceStart API was specified, the same profile must be specified here.

iIsNodeNum
Input. Indicates whether a node number is specified. Possible values are TRUE and FALSE. If specified, the stop command only affects the specified node.

iNodeNum
Input. The database partition number.

iStopOption
Input. Option. Valid values are:

SQLE_NONE
Issue the normal db2stop operation.

SQLE_FORCE
Issue the FORCE APPLICATION (ALL) command.

SQLE_DROP
Drop the node from the db2nodes.cfg file.

iCallerac
Input. This field is valid only for the SQLE_DROP value of the OPTION field. Valid values are:

SQLE_DROP
Initial call. This is the default value.

SQLE_CONTINUE
Subsequent call. Continue processing after a prompt.

SQLE_TERMINATE
Subsequent call. Terminate processing after a prompt.

db2gInstanceStopStruct data structure specific parameters

iRemoteInstLen
Input. Specifies the length in bytes of piRemoteInstName.

db2gDasCommData data structure specific parameters

iNodeOrHostNameLen
Input. Specifies the length in bytes of piNodeOrHostName.

iUserIdLen
Input. Specifies the length in bytes of piUserId.

iUserPwLen
Input. Specifies the length in bytes of piUserPw.

db2InstanceUnquiesce - Unquiesce instance

Unquiesce all databases in the instance.

Authorization

One of the following:

- sysadm
- sysctrl

Required connection

None

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2InstanceUnquiesce (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InsUnquiesceStruct
{
             char *piInstanceName;
} db2InsUnquiesceStruct;

SQL_API_RC SQL_API_FN
  db2gInstanceUnquiesce (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInsUnquiesceStruct
{
             db2Uint32 iInstanceNameLen;
             char *piInstanceName;
} db2gInsUnquiesceStruct;
```

db2InstanceUnquiesce API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2InsUnquiesceStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2InsUnquiesceStruct data structure parameters

piInstanceName
Input. The instance name.

db2gInsUnquiesceStruct data structure specific parameters

iInstanceNameLen
Input. Specifies the length in bytes of piInstanceName.

db2Load - Load data into a table

Loads data into a DB2 table. Data residing on the server may be in the form of a file, cursor, tape, or named pipe. Data residing on a remotely connected client may be in the form of a fully qualified file, a cursor, or named pipe. Although faster than the import utility, the load utility does not support loading data at the hierarchy level or loading into a nickname.

Authorization

One of the following:

- *dataaccess*
- load authority on the database and:
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.

If the FORCE option is specified, SYSADM authority is required.

Note: In general, all load processes and all DB2 server processes are owned by the instance owner. All of these processes use the identification of the instance owner to access needed files. Therefore, the instance owner must have read access to the input files, regardless of who invokes the command.

Required connection

Database. If implicit connect is enabled, a connection to the default database is established. Utility access to Linux, UNIX, or Windows database servers from Linux, UNIX, or Windows clients must be a direct connection through the engine and not through a DB2 Connect gateway or loop back environment.

Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

API include file

db2ApiDf.h

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Load (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LoadStruct
{
   struct sqlu_media_list *piSourceList;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piLocalMsgFileName;
   char *piTempFilesPath;
   struct sqlu_media_list *piVendorSortWorkPaths;
   struct sqlu_media_list *piCopyTargetList;
   db2int32 *piNullIndicators;
   struct db2LoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2PartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
   struct sqlu_media_list *piXmlPathList;
   struct sqllob *piLongActionString;
} db2LoadStruct;

typedef SQL_STRUCTURE db2LoadUserExit
{
   db2Char iSourceUserExitCmd;
   struct db2Char *piInputStream;
   struct db2Char *piInputFileName;
   struct db2Char *piOutputFileName;
   db2Uint16 *piEnableParallelism;
} db2LoadUserExit;

typedef SQL_STRUCTURE db2LoadIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   char *piUseTablespace;
   db2Uint32 iSavecount;
   db2Uint32 iDataBufferSize;
   db2Uint32 iSortBufferSize;
   db2Uint32 iWarningcount;
   db2Uint16 iHoldQuiesce;
   db2Uint16 iCpuParallelism;
   db2Uint16 iDiskParallelism;
   db2Uint16 iNonrecoverable;
   db2Uint16 iIndexingMode;
   db2Uint16 iAccessLevel;
   db2Uint16 iLockWithForce;
   db2Uint16 iCheckPending;
   char iRestartphase;
   char iStatsOpt;
   db2Uint16 *piXmlParse;
   db2DMUXmlValidate *piXmlValidate;
```

```
   db2Uint16 iSetIntegrityPending;
   struct db2LoadUserExit *piSourceUserExit;
} db2LoadIn;

typedef SQL_STRUCTURE db2LoadOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsLoaded;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsDeleted;
   db2Uint64 oRowsCommitted;
} db2LoadOut;

typedef SQL_STRUCTURE db2PartLoadIn
{
   char *piHostname;
   char *piFileTransferCmd;
   char *piPartFileLocation;
   struct db2LoadNodeList *piOutputNodes;
   struct db2LoadNodeList *piPartitioningNodes;
   db2Uint16 *piMode;
   db2Uint16 *piMaxNumPartAgents;
   db2Uint16 *piIsolatePartErrs;
   db2Uint16 *piStatusInterval;
   struct db2LoadPortRange *piPortRange;
   db2Uint16 *piCheckTruncation;
   char *piMapFileInput;
   char *piMapFileOutput;
   db2Uint16 *piTrace;
   db2Uint16 *piNewline;
   char *piDistfile;
   db2Uint16 *piOmitHeader;
   SQL_PDB_NODE_TYPE *piRunStatDBPartNum;
} db2PartLoadIn;

typedef SQL_STRUCTURE db2LoadNodeList
{
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint16 iNumNodes;
} db2LoadNodeList;

typedef SQL_STRUCTURE db2LoadPortRange
{
   db2Uint16 iPortMin;
   db2Uint16 iPortMax;
} db2LoadPortRange;

typedef SQL_STRUCTURE db2PartLoadOut
{
   db2Uint64 oRowsRdPartAgents;
   db2Uint64 oRowsRejPartAgents;
   db2Uint64 oRowsPartitioned;
   struct db2LoadAgentInfo *poAgentInfoList;
   db2Uint32 iMaxAgentInfoEntries;
   db2Uint32 oNumAgentInfoEntries;
} db2PartLoadOut;

typedef SQL_STRUCTURE db2LoadAgentInfo
{
   db2int32 oSqlcode;
   db2Uint32 oTableState;
   SQL_PDB_NODE_TYPE oNodeNum;
   db2Uint16 oAgentType;
} db2LoadAgentInfo;

SQL_API_RC SQL_API_FN
```

```
  db2gLoad (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gLoadStruct
{
   struct sqlu_media_list *piSourceList;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piLocalMsgFileName;
   char *piTempFilesPath;
   struct sqlu_media_list *piVendorSortWorkPaths;
   struct sqlu_media_list *piCopyTargetList;
   db2int32 *piNullIndicators;
   struct db2gLoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2gPartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
   db2Uint16 iFileTypeLen;
   db2Uint16 iLocalMsgFileLen;
   db2Uint16 iTempFilesPathLen;
   struct sqlu_media_list *piXmlPathList;
   struct sqllob *piLongActionString;
} db2gLoadStruct;

typedef SQL_STRUCTURE db2gLoadIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   char *piUseTablespace;
   db2Uint32 iSavecount;
   db2Uint32 iDataBufferSize;
   db2Uint32 iSortBufferSize;
   db2Uint32 iWarningcount;
   db2Uint16 iHoldQuiesce;
   db2Uint16 iCpuParallelism;
   db2Uint16 iDiskParallelism;
   db2Uint16 iNonrecoverable;
   db2Uint16 iIndexingMode;
   db2Uint16 iAccessLevel;
   db2Uint16 iLockWithForce;
   db2Uint16 iCheckPending;
   char iRestartphase;
   char iStatsOpt;
   db2Uint16 iUseTablespaceLen;
   db2Uint16 iSetIntegrityPending;
   db2Uint16 *piXmlParse;
   db2DMUXmlValidate *piXmlValidate;
   struct db2LoadUserExit *piSourceUserExit;
} db2gLoadIn;

typedef SQL_STRUCTURE db2gPartLoadIn
{
   char *piHostname;
   char *piFileTransferCmd;
   char *piPartFileLocation;
   struct db2LoadNodeList *piOutputNodes;
   struct db2LoadNodeList *piPartitioningNodes;
   db2Uint16 *piMode;
   db2Uint16 *piMaxNumPartAgents;
   db2Uint16 *piIsolatePartErrs;
   db2Uint16 *piStatusInterval;
```

```
   struct db2LoadPortRange *piPortRange;
   db2Uint16 *piCheckTruncation;
   char *piMapFileInput;
   char *piMapFileOutput;
   db2Uint16 *piTrace;
   db2Uint16 *piNewline;
   char *piDistfile;
   db2Uint16 *piOmitHeader;
   void *piReserved1;
   db2Uint16 iHostnameLen;
   db2Uint16 iFileTransferLen;
   db2Uint16 iPartFileLocLen;
   db2Uint16 iMapFileInputLen;
   db2Uint16 iMapFileOutputLen;
   db2Uint16 iDistfileLen;
} db2gPartLoadIn;


/* Definitions for iUsing value of db2DMUXmlValidate structure               */
#define DB2DMU_XMLVAL_XDS              1         /* Use XDS                  */
#define DB2DMU_XMLVAL_SCHEMA           2         /* Use a specified schema   */
#define DB2DMU_XMLVAL_SCHEMALOC_HINTS  3         /* Use schemaLocation hints */
#define DB2DMU_XMLVAL_ORIGSCHEMA       4         /* Use schema that document was
                                                    originally validated against
                                                    (load from cursor only) */
```

db2Load API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2LoadStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2LoadStruct data structure parameters

piSourceList
Input. A pointer to an sqlu_media_list structure used to provide a list of source files, devices, vendors, pipes, or SQL statements.

The information provided in this structure depends on the value of the media_type field. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_SQL_STMT
If the media_type field is set to this value, the caller provides an SQL query through the pStatement field of the target field. The pStatement field is of type sqlu_statement_entry. The sessions field must be set to the value of 1, since the load utility only accepts a single SQL query per load.

SQLU_SERVER_LOCATION
If the media_type field is set to this value, the caller provides information through sqlu_location_entry structures. The sessions field indicates the number of sqlu_location_entry structures provided. This is used for files, devices, and named pipes.

SQLU_CLIENT_LOCATION
If the media_type field is set to this value, the caller provides information through sqlu_location_entry structures. The sessions

field indicates the number of sqlu_location_entry structures provided. This is used for fully qualified files and named pipes. Note that this media_type is only valid if the API is being called via a remotely connected client.

SQLU_TSM_MEDIA
If the media_type field is set to this value, the sqlu_vendor structure is used, where filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of TSM sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

SQLU_OTHER_MEDIA
If the media_type field is set to this value, the sqlu_vendor structure is used, where shr_lib is the shared library name, and filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of other vendor sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

SQLU_REMOTEFETCH
If the media_type field is set to this value, the caller provides information through an sqlu_remotefetch_entry structure. The sessions field must be set to the value of 1.

piLobPathList
Input. A pointer to an sqlu_media_list structure. For IXF, ASC, and DEL file types, a list of fully qualified paths or devices to identify the location of the individual LOB files to be loaded. The file names are found in the IXF, ASC, or DEL files, and are appended to the paths provided.

The information provided in this structure depends on the value of the media_type field. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_LOCAL_MEDIA
If set to this value, the caller provides information through sqlu_media_entry structures. The sessions field indicates the number of sqlu_media_entry structures provided.

SQLU_TSM_MEDIA
If set to this value, the sqlu_vendor structure is used, where filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of TSM sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

SQLU_OTHER_MEDIA
If set to this value, the sqlu_vendor structure is used, where shr_lib is the shared library name, and filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of other vendor sessions to initiate. The load utility will

start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

piDataDescriptor
Input. Pointer to an sqldcol structure containing information about the columns being selected for loading from the external file.

If the piFileType parameter is set to SQL_ASC, the dcolmeth field of this structure must be set to SQL_METH_L. The user specifies the start and end locations for each column to be loaded.

If the file type is SQL_DEL, dcolmeth can be either SQL_METH_P or SQL_METH_D. If it is SQL_METH_P, the user must provide the source column position. If it is SQL_METH_D, the first column in the file is loaded into the first column of the table, and so on.

If the file type is SQL_IXF, dcolmeth can be one of SQL_METH_P, SQL_METH_D, or SQL_METH_N. The rules for DEL files apply here, except that SQL_METH_N indicates that file column names are to be provided in the sqldcol structure.

piActionString
Deprecated, replaced by piLongActionString.

piLongActionString
Input. Pointer to an *sqllob* structure containing a 4-byte long field, followed by an array of characters specifying an action that affects the table.

The character array is of the form:

```
"INSERT|REPLACE KEEPDICTIONARY|REPLACE RESETDICTIONARY|RESTART|TERMINATE
INTO tbname [(column_list)]
[FOR EXCEPTION e_tbname]"
```

INSERT
Adds the loaded data to the table without changing the existing table data.

REPLACE
Deletes all existing data from the table, and inserts the loaded data. The table definition and the index definitions are not changed.

RESTART
Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase.

TERMINATE
Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects may be marked as invalid, in which case index rebuild will automatically take place at next access). If the table spaces in which the table resides are not in load pending state, this option does not affect the state of the table spaces.

The load terminate option will not remove a backup pending state from table spaces.

tbname
The name of the table into which the data is to be loaded. The table cannot be a system table, a declared temporary table, or a created temporary table. An alias, or the fully qualified or

unqualified table name can be specified. A qualified table name is in the form schema.tablename. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

(column_list)
A list of table column names into which the data is to be inserted. The column names must be separated by commas. If a name contains spaces or lowercase characters, it must be enclosed by quotation marks.

FOR EXCEPTION e_tbname
Specifies the exception table into which rows in error will be copied. The exception table is used to store copies of rows that violate unique index rules, range constraints and security policies.

NORANGEEXC
Indicates that if a row is rejected because of a range violation it will not be inserted into the exception table.

NOUNIQUEEXC
Indicates that if a row is rejected because it violates a unique constraint it will not be inserted into the exception table.

piFileType
Input. A string that indicates the format of the input data source. Supported external formats (defined in sqlutil) are:

SQL_ASC
Non-delimited ASCII.

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_IXF
PC version of the Integration Exchange Format, the preferred method for exporting data from a table so that it can be loaded later into the same table or into another database manager table.

SQL_CURSOR
An SQL query. The sqlu_media_list structure passed in through the piSourceList parameter is either of type SQLU_SQL_STMT or SQLU_REMOTEFETCH, and refers to an SQL query or a table name.

piFileTypeMod
Input. A pointer to the sqlchar structure, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types. See related link ″File type modifiers for the load utility.″

piLocalMsgFileName
Input. A string containing the name of a local file to which output messages are to be written.

piTempFilesPath
Input. A string containing the path name to be used on the server for temporary files. Temporary files are created to store messages, consistency points, and delete phase information.

piVendorSortWorkPaths
Input. A pointer to the sqlu_media_list structure which specifies the Vendor Sort work directories.

piCopyTargetList
Input. A pointer to an sqlu_media_list structure used (if a copy image is to be created) to provide a list of target paths, devices, or a shared library to which the copy image is to be written.

The values provided in this structure depend on the value of the media_type field. Valid values for this parameter (defined in sqlutil header file, located in the include directory) are:

SQLU_LOCAL_MEDIA
If the copy is to be written to local media, set the media_type to this value and provide information about the targets in sqlu_media_entry structures. The sessions field specifies the number of sqlu_media_entry structures provided.

SQLU_TSM_MEDIA
If the copy is to be written to TSM, use this value. No further information is required.

SQLU_OTHER_MEDIA
If a vendor product is to be used, use this value and provide further information via an sqlu_vendor structure. Set the shr_lib field of this structure to the shared library name of the vendor product. Provide only one sqlu_vendor entry, regardless of the value of sessions. The sessions field specifies the number of sqlu_media_entry structures provided. The load utility will start the sessions with different sequence numbers, but with the same data provided in the one sqlu_vendor entry.

piNullIndicators
Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. There is a one-to-one ordered correspondence between the elements of this array and the columns being loaded from the data file. That is, the number of elements must equal the dcolnum field of the piDataDescriptor parameter. Each element of the array contains a number identifying a location in the data file that is to be used as a NULL indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified location in the data file must contain a Y or an N. A Y indicates that the table column data is NULL, and N indicates that the table column data is not NULL.

piLoadInfoIn
Input. A pointer to the db2LoadIn structure.

poLoadInfoOut
Output. A pointer to the db2LoadOut structure.

piPartLoadInfoIn
Input. A pointer to the db2PartLoadIn structure.

poPartLoadInfoOut
Output. A pointer to the db2PartLoadOut structure.

iCallerAction

Input. An action requested by the caller. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_INITIAL
Initial call. This value (or SQLU_NOINTERRUPT) must be used on the first call to the API.

SQLU_NOINTERRUPT
Initial call. Do not suspend processing. This value (or SQLU_INITIAL) must be used on the first call to the API.

If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested load operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD_PENDING state. This option should be specified if further processing of the data is not to be done.

SQLU_ABORT
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD_PENDING state. This option should be specified if further processing of the data is not to be done.

SQLU_RESTART
Restart processing.

SQLU_DEVICE_TERMINATE
Terminate a single device. This option should be specified if the utility is to stop reading data from the device, but further processing of the data is to be done.

piXmlPathList

Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the xml files can be found.

db2LoadUserExit data structure parameters

iSourceUserExitCmd

Input. The fully qualified name of an executable that will be used to feed data to the utility. For security reasons, the executable must be placed within the `sqllib/bin` directory on the server. This parameter is mandatory if the piSourceUserExit structure is not NULL.

The piInputStream, piInputFileName, piOutputFileName and piEnableParallelism fields are optional.

piInputStream
Input. A generic byte-stream that will be passed directly to the user-exit application via STDIN. You have complete control over what data is contained in this byte-stream and in what format. The load utility will simply carry this byte-stream over to the server and pass it into the user-exit application by feeding the process' STDIN (it will not codepage convert or modify the byte-stream). Your user-exit application would read the arguments from STDIN and use the data however intended.

One important attribute of this feature is the ability to hide sensitive information (such as userid/passwords).

piInputFileName
Input. Contains the name of a fully qualified client-side file, whose contents will be passed into the user-exit application by feeding the process' STDIN.

piOutputFileName
Input. The fully qualified name of a server-side file. The STDOUT and STDERR streams of the process which is executing the user-exit application will be streamed into this file. When piEnableParallelism is TRUE, multiple files will be created (one per user-exit instance), and each file name will be appended with a 3 digit numeric node-number value, such as `<filename>.000`).

piEnableParallelism
Input. A flag indicating that the utility should attempt to parallelize the invocation of the user-exit application.

db2LoadIn data structure parameters

iRowcount
Input. The number of physical records to be loaded. Allows a user to load only the first rowcnt rows in a file.

iRestartcount
Input. Reserved for future use.

piUseTablespace
Input. If the indexes are being rebuilt, a shadow copy of the index is built in table space iUseTablespaceName and copied over to the original table space at the end of the load. Only system temporary table spaces can be used with this option. If not specified then the shadow index will be created in the same table space as the index object.

If the shadow copy is created in the same table space as the index object, the copy of the shadow index object over the old index object is instantaneous. If the shadow copy is in a different table space from the index object a physical copy is performed. This could involve considerable I/O and time. The copy happens while the table is offline at the end of a load.

This field is ignored if iAccessLevel is SQLU_ALLOW_NO_ACCESS.

This option is ignored if the user does not specify INDEXING MODE REBUILD or INDEXING MODE AUTOSELECT. This option will also be ignored if INDEXING MODE AUTOSELECT is chosen and load chooses to incrementally update the index.

iSavecount
The number of records to load before establishing a consistency point. This

value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using db2LoadQuery - Load Query. If the value of savecount is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance.

The default value is 0, meaning that no consistency points will be established, unless necessary.

iDataBufferSize
The number of 4KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the required minimum is used, and no warning is returned.

This memory is allocated directly from the utility heap, whose size can be modified through the util_heap_sz database configuration parameter.

If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

iSortBufferSize
Input. This option specifies a value that overrides the SORTHEAP database configuration parameter during a load operation. It is relevant only when loading tables with indexes and only when the iIndexingMode parameter is not specified as SQLU_INX_DEFERRED. The value that is specified cannot exceed the value of SORTHEAP. This parameter is useful for throttling the sort memory used by LOAD without changing the value of SORTHEAP, which would also affect general query processing.

iWarningcount
Input. Stops the load operation after warningcnt warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the load file or the target table is specified incorrectly, the load utility will generate a warning for each row that it attempts to load, which will cause the load to fail. If warningcnt is 0, or this option is not specified, the load operation will continue regardless of the number of warnings issued.

If the load operation is stopped because the threshold of warnings was exceeded, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

iHoldQuiesce
Input. A flag whose value is set to TRUE if the utility is to leave the table in quiesced exclusive state after the load, and to FALSE if it is not.

iCpuParallelism
Input. The number of processes or threads that the load utility will create for parsing, converting and formatting records when building table objects. This parameter is designed to exploit intra-partition parallelism. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, the load utility uses an intelligent default value at run time. Note: If this parameter

is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs, or the value specified by the user.

iDiskParallelism
Input. The number of processes or threads that the load utility will create for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

iNonrecoverable
Input. Set to SQLU_NON_RECOVERABLE_LOAD if the load transaction is to be marked as non-recoverable, and it will not be possible to recover it by a subsequent roll forward action. The rollforward utility will skip the transaction, and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the roll forward is completed, such a table can only be dropped. With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation. Set to SQLU_RECOVERABLE_LOAD if the load transaction is to be marked as recoverable.

iIndexingMode
Input. Specifies the indexing mode. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_INX_AUTOSELECT
LOAD chooses between REBUILD and INCREMENTAL indexing modes.

SQLU_INX_REBUILD
Rebuild table indexes.

SQLU_INX_INCREMENTAL
Extend existing indexes.

SQLU_INX_DEFERRED
Do not update table indexes.

iAccessLevel
Input. Specifies the access level. Valid values are:

SQLU_ALLOW_NO_ACCESS
Specifies that the load locks the table exclusively.

SQLU_ALLOW_READ_ACCESS
Specifies that the original data in the table (the non-delta portion) should still be visible to readers while the load is in progress. This option is only valid for load appends, such as a load insert, and will be ignored for load replace.

iLockWithForce
Input. A boolean flag. If set to TRUE load will force other applications as necessary to ensure that it obtains table locks immediately. This option requires the same authority as the FORCE APPLICATIONS command (SYSADM or SYSCTRL).

SQLU_ALLOW_NO_ACCESS loads may force conflicting applications at the start of the load operation. At the start of the load, the utility may force applications that are attempting to either query or modify the table.

SQLU_ALLOW_READ_ACCESS loads may force conflicting applications at the start or end of the load operation. At the start of the load, the load utility may force applications that are attempting to modify the table. At the end of the load, the load utility may force applications that are attempting to either query or modify the table.

iCheckPending
This parameter is being deprecated as of Version 9.1. Use the iSetIntegrityPending parameter instead.

iRestartphase
Input. Reserved. Valid value is a single space character ' '.

iStatsOpt
Input. Granularity of statistics to collect. Valid values are:

SQLU_STATS_NONE
No statistics to be gathered.

SQLU_STATS_USE_PROFILE
Statistics are collected based on the profile defined for the current table. This profile must be created using the RUNSTATS command. If no profile exists for the current table, a warning is returned and no statistics are collected.

iSetIntegrityPending
Input. Specifies to put the table into set integrity pending state. If the value SQLU_SI_PENDING_CASCADE_IMMEDIATE is specified, set integrity pending state will be immediately cascaded to all dependent and descendent tables. If the value SQLU_SI_PENDING_CASCADE_DEFERRED is specified, the cascade of set integrity pending state to dependent tables will be deferred until the target table is checked for integrity violations.
SQLU_SI_PENDING_CASCADE_DEFERRED is the default value if the option is not specified.

piSourceUserExit
Input. A pointer to the db2LoadUserExit structure.

piXmlParse
Input. Type of parsing that should occur for XML documents. Valid values found in the db2ApiDf header file in the include directory are:

DB2DMU_XMLPARSE_PRESERVE_WS
Whitespace should be preserved.

DB2DMU_XMLPARSE_STRIP_WS
Whitespace should be stripped.

piXmlValidate
Input. Pointer to the db2DMUXmlValidate structure. Indicates that XML schema validation should occur for XML documents.

```
/* XML Validate structure                                          */
typedef SQL_STRUCTURE db2DMUXmlValidate
{
   db2Uint16                        iUsing;      /* What to use to perform   */
                                                 /* validation               */
   struct db2DMUXmlValidateXds    *piXdsArgs;    /* Arguments for            */
                                                 /* XMLVALIDATE USING XDS    */
   struct db2DMUXmlValidateSchema *piSchemaArgs; /* Arguments for            */
                                                 /* XMLVALIDATE USING SCHEMA */
} db2DMUXmlValidate;
```

db2LoadOut data structure parameters

oRowsRead
Output. Number of records read during the load operation.

oRowsSkipped
Output. Number of records skipped before the load operation begins.

oRowsLoaded
Output. Number of rows loaded into the target table.

oRowsRejected
Output. Number of records that could not be loaded.

oRowsDeleted
Output. Number of duplicate rows deleted.

oRowsCommitted
Output. The total number of processed records: the number of records loaded successfully and committed to the database, plus the number of skipped and rejected records.

db2PartLoadIn data structure parameters

piHostname
Input. The hostname for the iFileTransferCmd parameter. If NULL, the hostname will default to ″nohost″. This parameter is deprecated.

piFileTransferCmd
Input. File transfer command parameter. If not required, it must be set to NULL. This parameter is deprecated. Use the piSourceUserExit parameter instead.

piPartFileLocation
Input. In PARTITION_ONLY, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, this parameter can be used to specify the location of the partitioned files. This location must exist on each database partition specified by the piOutputNodes option.

For the SQL_CURSOR file type, this parameter cannot be NULL and the location does not refer to a path, but to a fully qualified file name. This will be the fully qualified base file name of the partitioned files that are created on each output database partition for PARTITION_ONLY mode, or the location of the files to be read from each database partition for LOAD_ONLY mode. For PARTITION_ONLY mode, multiple files may be created with the specified base name if there are LOB columns in the target table. For file types other than SQL_CURSOR, if the value of this parameter is NULL, it will default to the current directory.

piOutputNodes
Input. The list of Load output database partitions. A NULL indicates that all nodes on which the target table is defined.

piPartitioningNodes
Input. The list of partitioning nodes. A NULL indicates the default.

piMode
Input. Specifies the load mode for partitioned databases. Valid values (defined in db2ApiDf header file, located in the include directory) are:

- DB2LOAD_PARTITION_AND_LOAD
Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

- DB2LOAD_PARTITION_ONLY
Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. For file types other than SQL_CURSOR, the name of the output file on each database partition will have the form filename.xxx, where filename is the name of the first input file specified by piSourceList and xxx is the database partition number.For the SQL_CURSOR file type, the name of the output file on each database partition will be determined by the piPartFileLocation parameter. Refer to the piPartFileLocation parameter for information about how to specify the location of the database partition file on each database partition.

Note: This mode cannot be used for a CLI LOAD.

DB2LOAD_LOAD_ONLY
Data is assumed to be already distributed; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. For file types other than SQL_CURSOR, the input file name on each database partition is expected to be of the form filename.xxx, where filename is the name of the first file specified by piSourceList and xxx is the 13-digit database partition number. For the SQL_CURSOR file type, the name of the input file on each database partition will be determined by the piPartFileLocation parameter. Refer to the piPartFileLocation parameter for information about how to specify the location of the database partition file on each database partition.

Note: This mode cannot be used when loading a data file located on a remote client, nor can it be used for a CLI LOAD.

DB2LOAD_LOAD_ONLY_VERIFY_PART
Data is assumed to be already distributed, but the data file does not contain a database partition header. The distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dumpfile if the dumpfile file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning will be written to the load message file for that database partition. The input file name on each database partition is expected to be of the form filename.xxx, where filename is the name of the first file specified by piSourceList and xxx is the 13-digit database partition number.

Note: This mode cannot be used when loading a data file located on a remote client, nor can it be used for a CLI LOAD.

DB2LOAD_ANALYZE
An optimal distribution map with even distribution across all database partitions is generated.

piMaxNumPartAgents
Input. The maximum number of partitioning agents. A NULL value indicates the default, which is 25.

piIsolatePartErrs

Input. Indicates how the load operation will react to errors that occur on individual database partitions. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOAD_SETUP_ERRS_ONLY

In this mode, errors that occur on a database partition during setup, such as problems accessing a database partition or problems accessing a table space or table on a database partition, will cause the load operation to stop on the failing database partitions but to continue on the remaining database partitions. Errors that occur on a database partition while data is being loaded will cause the entire operation to fail and rollback to the last point of consistency on each database partition.

DB2LOAD_LOAD_ERRS_ONLY

In this mode, errors that occur on a database partition during setup will cause the entire load operation to fail. When an error occurs while data is being loaded, the database partitions with errors will be rolled back to their last point of consistency. The load operation will continue on the remaining database partitions until a failure occurs or until all the data is loaded. On the database partitions where all of the data was loaded, the data will not be visible following the load operation. Because of the errors in the other database partitions the transaction will be aborted. Data on all of the database partitions will remain invisible until a load restart operation is performed. This will make the newly loaded data visible on the database partitions where the load operation completed and resume the load operation on database partitions that experienced an error.

Note: This mode cannot be used when iAccessLevel is set to SQLU_ALLOW_READ_ACCESS and a copy target is also specified.

DB2LOAD_SETUP_AND_LOAD_ERRS

In this mode, database partition-level errors during setup or loading data cause processing to stop only on the affected database partitions. As with the DB2LOAD_LOAD_ERRS_ONLY mode, when database partition errors do occur while data is being loaded, the data on all database partitions will remain invisible until a load restart operation is performed.

Note: This mode cannot 1be used when iAccessLevel is set to SQLU_ALLOW_READ_ACCESS and a copy target is also specified.

DB2LOAD_NO_ISOLATION

Any error during the Load operation causes the transaction to be aborted. If this parameter is NULL, it will default to DB2LOAD_LOAD_ERRS_ONLY, unless iAccessLevel is set to SQLU_ALLOW_READ_ACCESS and a copy target is also specified, in which case the default is DB2LOAD_NO_ISOLATION.

piStatusInterval

Input. Specifies the number of megabytes (MB) of data to load before generating a progress message. Valid values are whole numbers in the range of 1 to 4000. If NULL is specified, a default value of 100 will be used.

piPortRange
Input. The TCP port range for internal communication. If NULL, the port range used will be 6000-6063.

piCheckTruncation
Input. Causes Load to check for record truncation at Input/Output. Valid values are TRUE and FALSE. If NULL, the default is FALSE.

piMapFileInput
Input. Distribution map input filename. If the mode is not ANALYZE, this parameter should be set to NULL. If the mode is ANALYZE, this parameter must be specified.

piMapFileOutput
Input. Distribution map output filename. The rules for piMapFileInput apply here as well.

piTrace
Input. Specifies the number of records to trace when you need to review a dump of all the data conversion process and the output of hashing values. If NULL, the number of records defaults to 0.

piNewline
Input. Forces Load to check for newline characters at end of ASC data records if RECLEN file type modifier is also specified. Possible values are TRUE and FALSE. If NULL, the value defaults to FALSE.

piDistfile
Input. Name of the database partition distribution file. If a NULL is specified, the value defaults to ″DISTFILE″.

piOmitHeader
Input. Indicates that a distribution map header should not be included in the database partition file when using DB2LOAD_PARTITION_ONLY mode. Possible values are TRUE and FALSE. If NULL, the default is FALSE.

piRunStatDBPartNum
Specifies the database partition on which to collect statistics. The default value is the first database partition in the output database partition list.

db2LoadNodeList data structure parameters

piNodeList
Input. An array of node numbers.

iNumNodes
Input. The number of nodes in the piNodeList array. A 0 indicates the default, which is all nodes on which the target table is defined.

db2LoadPortRange data structure parameters

iPortMin
Input. Lower port number.

iPortMax
Input. Higher port number.

db2PartLoadOut data structure parameters

oRowsRdPartAgents
Output. Total number of rows read by all partitioning agents.

oRowsRejPartAgents
Output. Total number of rows rejected by all partitioning agents.

oRowsPartitioned
Output. Total number of rows partitioned by all partitioning agents.

poAgentInfoList
Output. During a load operation into a partitioned database, the following load processing entities may be involved: load agents, partitioning agents, pre-partitioning agents, file transfer command agents and load-to-file agents (these are described in the Data Movement Guide). The purpose of the poAgentInfoList output parameter is to return to the caller information about each load agent that participated in a load operation. Each entry in the list contains the following information:

oAgentType
A tag indicating what kind of load agent the entry describes.

oNodeNum
The number of the database partition on which the agent executed.

oSqlcode
The final sqlcode resulting from the agent's processing.

oTableState
The final status of the table on the database partition on which the agent executed (relevant only for load agents).

It is up to the caller of the API to allocate memory for this list prior to calling the API. The caller should also indicate the number of entries for which they allocated memory in the iMaxAgentInfoEntries parameter. If the caller sets poAgentInfoList to NULL or sets iMaxAgentInfoEntries to 0, then no information will be returned about the load agents.

iMaxAgentInfoEntries
Input. The maximum number of agent information entries allocated by the user for poAgentInfoList. In general, setting this parameter to 3 times the number of database partitions involved in the load operation should be sufficient.

oNumAgentInfoEntries
Output. The actual number of agent information entries produced by the load operation. This number of entries will be returned to the user in the poAgentInfoList parameter as long as iMaxAgentInfoEntries is greater than or equal to oNumAgentInfoEntries. If iMaxAgentInfoEntries is less than oNumAgentInfoEntries, then the number of entries returned in poAgentInfoList is equal to iMaxAgentInfoEntries.

db2LoadAgentInfo data structure parameters

oSqlcode
Output. The final sqlcode resulting from the agent's processing.

oTableState
Output. The purpose of this output parameter is not to report every possible state of the table after the load operation. Rather, its purpose is to report only a small subset of possible tablestates in order to give the caller a general idea of what happened to the table during load processing. This value is relevant for load agents only. The possible values are:

DB2LOADQUERY_NORMAL
Indicates that the load completed successfully on the database

partition and the table was taken out of the LOAD IN PROGRESS (or LOAD PENDING) state. In this case, the table still could be in SET INTEGRITY PENDING state due to the need for further constraints processing, but this will not reported as this is normal.

DB2LOADQUERY_UNCHANGED
Indicates that the load job aborted processing due to an error but did not yet change the state of the table on the database partition from whatever state it was in prior to calling db2Load. It is not necessary to perform a load restart or terminate operation on such database partitions.

DB2LOADQUERY_LOADPENDING
Indicates that the load job aborted during processing but left the table on the database partition in the LOAD PENDING state, indicating that the load job on that database partition must be either terminated or restarted.

oNodeNum
Output. The number of the database partition on which the agent executed.

oAgentType
Output. The agent type. Valid values (defined in db2ApiDf header file, located in the include directory) are :

- DB2LOAD_LOAD_AGENT
- DB2LOAD_PARTITIONING_AGENT
- DB2LOAD_PRE_PARTITIONING_AGENT
- DB2LOAD_FILE_TRANSFER_AGENT
- DB2LOAD_LOAD_TO_FILE_AGENT

db2gLoadStruct data structure specific parameters

iFileTypeLen
Input. Specifies the length in bytes of iFileType parameter.

iLocalMsgFileLen
Input. Specifies the length in bytes of iLocalMsgFileName parameter.

iTempFilesPathLen
Input. Specifies the length in bytes of iTempFilesPath parameter.

piXmlPathList
Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the xml files can be found.

db2gLoadIn data structure specific parameters

iUseTablespaceLen
Input. The length in bytes of piUseTablespace parameter.

piXmlParse
Input. Type of parsing that should occur for XML documents. Valid values found in the db2ApiDf header file in the include directory are:

DB2DMU_XMLPARSE_PRESERVE_WS
Whitespace should be preserved.

DB2DMU_XMLPARSE_STRIP_WS
Whitespace should be stripped.

piXmlValidate
Input. Pointer to the db2DMUXmlValidate structure. Indicates that XML schema validation should occur for XML documents.

```
/* XML Validate structure                                              */
typedef SQL_STRUCTURE db2DMUXmlValidate
{
   db2Uint16                       iUsing;       /* What to use to perform   */
                                                 /* validation               */
   struct db2DMUXmlValidateXds    *piXdsArgs;    /* Arguments for            */
                                                 /* XMLVALIDATE USING XDS    */
   struct db2DMUXmlValidateSchema *piSchemaArgs; /* Arguments for            */
                                                 /* XMLVALIDATE USING SCHEMA */
} db2DMUXmlValidate;
```

db2gPartLoadIn data structure specific parameters

piReserved1
Reserved for future use.

iHostnameLen
Input. The length in bytes of piHostname parameter.

iFileTransferLen
Input. The length in bytes of piFileTransferCmd parameter.

iPartFileLocLen
Input. The length in bytes of piPartFileLocation parameter.

iMapFileInputLen
Input. The length in bytes of piMapFileInput parameter.

iMapFileOutputLen
Input. The length in bytes of piMapFileOutput parameter.

iDistfileLen
Input. The length in bytes of piDistfile parameter.

Usage notes

Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a load is attempted.

The load utility builds indexes based on existing definitions. The exception tables are used to handle duplicates on unique keys. The utility does not enforce referential integrity, perform constraints checking, or update summary tables that are dependent on the tables being loaded. Tables that include referential or check constraints are placed in set integrity pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in set integrity pending state. Issue the SET INTEGRITY statement to take the tables out of set integrity pending state. Load operations cannot be carried out on replicated summary tables.

For clustering indexes, the data should be sorted on the clustering index prior to loading. The data need not be sorted when loading into an multi-dimensionally clustered (MDC) table.

db2Recover - Restore and roll forward a database

Restores and rolls forward a database to a particular point in time or to the end of the logs.

Scope

In a partitioned database environment, this API can only be called from the catalog partition. If no database partition servers are specified, it affects all database partition servers that are listed in the db2nodes.cfg file. If a point in time is specified, the API affects all database partitions.

Authorization

To recover an existing database, one of the following:

- sysadm
- sysctrl
- sysmaint

To recover to a new database, one of the following:

- sysadm
- sysctrl

Required connection

To recover an existing database, a database connection is required. This API automatically establishes a connection to the specified database and will release the connection when the recover operation finishes. Instance and database, to recover to a new database. The instance attachment is required to create the database.

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Recover (
       db2Uint32 versionNumber,
       void * pDB2RecovStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RecoverStruct
{
  char *piSourceDBAlias;
  char *piUsername;
  char *piPassword;
  db2Uint32 iRecoverCallerAction;
  db2Uint32 iOptions;
  sqlint32 *poNumReplies;
  struct sqlurf_info *poNodeInfo;
  char *piStopTime;
  char *piOverflowLogPath;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piHistoryFile;
  db2Uint32 iNumChngHistoryFile;
  struct sqlu_histFile *piChngHistoryFile;
  char *piComprLibrary;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2RecoverStruct;
```

```
SQL_STRUCTURE sqlu_histFile
{
   SQL_PDB_NODE_TYPE nodeNum;
   unsigned short  filenameLen;
   char filename[SQL_FILENAME_SZ+1];
};

SQL_API_RC SQL_API_FN
  db2gRecover (
       db2Uint32 versionNumber,
       void * pDB2gRecoverStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRecoverStruct
{
  char *piSourceDBAlias;
  db2Uint32 iSourceDBAliasLen;
  char *piUserName;
  db2Uint32 iUserNameLen;
  char *piPassword;
  db2Uint32 iPasswordLen;
  db2Uint32 iRecoverCallerAction;
  db2Uint32 iOptions;
  sqlint32 *poNumReplies;
  struct sqlurf_info *poNodeInfo;
  char *piStopTime;
  db2Uint32 iStopTimeLen;
  char *piOverflowLogPath;
  db2Uint32 iOverflowLogPathLen;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piHistoryFile;
  db2Uint32 iHistoryFileLen;
  db2Uint32 iNumChngHistoryFile;
  struct sqlu_histFile *piChngHistoryFile;
  char *piComprLibrary;
  db2Uint32 iComprLibraryLen;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2gRecoverStruct;
```

db2Recover API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pDB2RecoverStruct.

pDB2RecoverStruct
Input. A pointer to the db2RecoverStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2RecoverStruct data structure parameters

piSourceDBAlias
Input. A string containing the database alias of the database to be recovered.

piUserName
Input. A string containing the user name to be used when attempting a connection. Can be NULL.

piPassword
Input. A string containing the password to be used with the user name. Can be NULL.

iRecoverCallerAction
Input. Valid values are:

DB2RECOVER
Starts the recover operation. Specifies that the recover will run unattended, and that scenarios that normally require user intervention will either be attempted without first returning to the caller, or will generate an error. Use this caller action, for example, if it is known that all of the media required for the recover have been mounted, and utility prompts are not desired.

DB2RECOVER_RESTART
Allows the user to ignore a prior recover and start over from the beginning.

DB2RECOVER_CONTINUE
Continue using the device that generated the warning message (for example, when a new tape has been mounted).

DB2RECOVER_LOADREC_TERM
Terminate all devices being used by load recovery.

DB2RECOVER_DEVICE_TERM
Stop using the device that generated the warning message (for example, when there are no more tapes).

DB2RECOVER_PARM_CHK_ONLY
Used to validate parameters without performing a recover operation. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.

DB2RECOVER_DEVICE_TERMINATE
Removes a particular device from the list of devices used by the recover operation. When a particular device has exhausted its input, recover will return a warning to the caller. Call the recover utility again with this caller action to remove the device that generated the warning from the list of devices being used.

iOptions
Input. Valid values are:

- DB2RECOVER_EMPTY_FLAG
No flags specified.

- DB2RECOVER_LOCAL_TIME
Indicates that the value specified for the stop time by piStopTime is in local time, not GMT. This is the default setting.

- DB2RECOVER_GMT_TIME
This flag indicates that the value specified for the stop time by piStopTime is in GMT (Greenwich Mean Time).

poNumReplies
Output. The number of replies received.

poNodeInfo
Output. Database partition reply information.

piStopTime
Input. A character string containing a time stamp in ISO format. Database recovery will stop when this time stamp is exceeded. Specify SQLUM_INFINITY_TIMESTAMP to roll forward as far as possible. May be NULL for DB2ROLLFORWARD_QUERY, DB2ROLLFORWARD_PARM_CHECK, and any of the load recovery (DB2ROLLFORWARD_LOADREC_) caller actions.

piOverflowLogPath
Input. This parameter is used to specify an alternate log path to be used. In addition to the active log files, archived log files need to be moved (by the user) into the location specified by the logpath configuration parameter before they can be used by this utility. This can be a problem if the user does not have sufficient space in the log path. The overflow log path is provided for this reason. During roll-forward recovery, the required log files are searched, first in the log path, and then in the overflow log path. The log files needed for table space rollforward recovery can be brought into either the log path or the overflow log path. If the caller does not specify an overflow log path, the default value is the log path.

In a partitioned database environment, the overflow log path must be a valid, fully qualified path; the default path is the default overflow log path for each database partition. In a single-partition database environment, the overflow log path can be relative if the server is local.

iNumChngLgOvrflw
Input. Partitioned database environments only. The number of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

piChngLogOvrflw
Input. Partitioned database environments only. A pointer to a structure containing the fully qualified names of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

iAllNodeFlag
Input. Partitioned database environments only. Indicates whether the rollforward operation is to be applied to all database partition servers defined in db2nodes.cfg. Valid values are:

DB2_NODE_LIST
Apply to database partition servers in a list that is passed in piNodeList.

DB2_ALL_NODES
Apply to all database partition servers. piNodeList should be NULL. This is the default value.

DB2_ALL_EXCEPT
Apply to all database partition servers except those in a list that is passed in piNodeList.

DB2_CAT_NODE_ONLY
Apply to the catalog partition only. piNodeList should be NULL.

iNumNodes
Input. Specifies the number of database partition servers in the piNodeList array.

piNodeList
Input. A pointer to an array of database partition server numbers on which to perform the rollforward recovery.

iNumNodeInfo
Input. Defines the size of the output parameter poNodeInfo, which must be large enough to hold status information from each database partition that is being rolled forward. In a single-partition database environment, this parameter should be set to 1. The value of this parameter should be the same as the number of database partition servers for which this API is being called.

piHistoryFile
History file.

iNumChngHistoryFile
Number of history files in list.

piChngHistoryFile
List of history files.

piComprLibrary
Input. Indicates the name of the external library to be used to perform decompression of the backup image if the image is compressed. The name must be a fully-qualified path referring to a file on the server. If the value is a null pointer or a pointer to an empty string, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library is not found, the restore will fail.

piComprOptions
Input. Describes a block of binary data that will be passed to the initialization routine in the decompression library. DB2 will pass this string directly from the client to the server, so any issues of byte-reversal or code-page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of piComprOptions and iComprOptionsSize with the contents and size of this file respectively and will pass these new values to the initialization routine instead.

iComprOptionsSize
Input. Represents the size of the block of data passed as piComprOptions. iComprOptionsSize shall be zero if and only if piComprOptions is a null pointer.

sqlu_histFile data structure parameters

nodeNum
Input. Specifies which database partition this entry should be used for.

filenameLen
Input. Length in bytes of filename.

filename
Input. Path to the history file for this database partition. The path must end with a slash.

db2gRecoverStruct data structure specific parameters

iSourceDBAliasLen
Specifies the length in bytes of the piSourceDBAlias parameter.

iUserNameLen
Specified the length in bytes of the piUsername parameter.

iPasswordLen
Specifies the length in bytes of the piPassword parameter.

iStopTimeLen
Specifies the length in bytes of the piStopTime parameter.

iOverflowLogPathLen
Specifies the length in bytes of the piOverflowLogPath parameter.

iHistoryFileLen
Specifies the length in bytes of the piHistoryFile parameter.

iComprLibraryLen
Input. Specifies the length in bytes of the name of the library specified in the piComprLibrary parameter. Set to zero if no library name is given.

db2Reorg - Reorganize an index or a table

Reorganizes a table or all indexes defined on a table by compacting the information and reconstructing the rows or index data to eliminate fragmented data.

Authorization

One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT
- DBADM
- SQLADM
- CONTROL privilege on the table

Required connection

Database

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Reorg (
             db2Uint32 versionNumber,
             void * pReorgStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReorgStruct
{
  db2Uint32 reorgType;
  db2Uint32 reorgFlags;
  db2int32 nodeListFlag;
```

```
  db2Uint32 numNodes;
  SQL_PDB_NODE_TYPE *pNodeList;
  union db2ReorgObject        reorgObject;
} db2ReorgStruct;

union db2ReorgObject
{
  struct db2ReorgTable             tableStruct;
  struct db2ReorgIndexesAll        indexesAllStruct;
};

typedef SQL_STRUCTURE db2ReorgTable
{
  char *pTableName;
  char *pOrderByIndex;
  char *pSysTempSpace;
  char *pLongTempSpace;
  char *pPartitionName;
} db2ReorgTable;

typedef SQL_STRUCTURE db2ReorgIndexesAll
{
  char *pTableName;
  char *pIndexName;
  char *pPartitionName;
} db2ReorgIndexesAll;

SQL_API_RC SQL_API_FN
  db2gReorg (
              db2Uint32 versionNumber,
              void * pReorgStruct,
              struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gReorgStruct
{
  db2Uint32 reorgType;
  db2Uint32 reorgFlags;
  db2int32 nodeListFlag;
  db2Uint32 numNodes;
  SQL_PDB_NODE_TYPE *pNodeList;
  union db2gReorgObject       reorgObject;
} db2gReorgStruct;

typedef SQL_STRUCTURE db2gReorgNodes
{
  SQL_PDB_NODE_TYPE nodeNum[SQL_PDB_MAX_NUM_NODE];
} db2gReorgNodes;

union db2gReorgObject
{
  struct db2gReorgTable            tableStruct;
  struct db2gReorgIndexesAll       indexesAllStruct;
};

typedef SQL_STRUCTURE db2gReorgTable
{
  db2Uint32 tableNameLen;
  char *pTableName;
  db2Uint32 orderByIndexLen;
  char *pOrderByIndex;
  db2Uint32 sysTempSpaceLen;
  char *pSysTempSpace;
  db2Uint32 longTempSpaceLen;
  char *pLongTempSpace;
  db2Uint32 partitionNameLen;
  char *pPartitionName;
} db2gReorgTable;
```

```
typedef SQL_STRUCTURE db2gReorgIndexesAll
{
  db2Uint32 tableNameLen;
  char *pTableName;
  db2Uint32 indexNameLen;
  char *pIndexName;
  db2Uint32 indexNameLen;
  char *pPartitionName;
} db2gReorgIndexesAll;
```

db2Reorg API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter, **pReorgStruct**.

pReorgStruct
Input. A pointer to the db2ReorgStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ReorgStruct data structure parameters

reorgType
Input. Specifies the type of reorganization. Valid values (defined in `db2ApiDf` header file, located in the `include` directory) are:

DB2REORG_OBJ_TABLE_OFFLINE
Reorganize the table offline.

DB2REORG_OBJ_TABLE_INPLACE
Reorganize the table inplace.

DB2REORG_OBJ_INDEXESALL
Reorganize all indexes.

DB2REORG_OBJ_INDEX
Reorganize one index.

DB2REORG_RECLAIM_EXTENTS
Reorganize a multidimensional clustering (MDC) table to reclaim empty extents for the table space.

reorgFlags
Input. Reorganization options. Valid values (defined in `db2ApiDf` header file, located in the `include` directory) are:

DB2REORG_OPTION_NONE
Default action.

DB2REORG_LONGLOB
Reorganize long fields and lobs, used when DB2REORG_OBJ_TABLE_OFFLINE is specified as the **reorgType**. If DB2REORG_RESETDICTIONARY or DB2REORG_KEEPDICTIONARY option is also specified, the options apply to the XML storage object of the table in addition to the table object.

DB2REORG_INDEXSCAN
Recluster utilizing index scan, used when DB2REORG_OBJ_TABLE_OFFLINE is specified as the **reorgType**.

DB2REORG_START_ONLINE
Start online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the **reorgType**.

DB2REORG_PAUSE_ONLINE
Pause an existing online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the **reorgType**.

DB2REORG_STOP_ONLINE
Stop an existing online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the **reorgType**.

DB2REORG_RESUME_ONLINE
Resume a paused online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the **reorgType**.

DB2REORG_NOTRUNCATE_ONLINE
Do not perform table truncation, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the **reorgType**.

DB2REORG_ALLOW_NONE
No read or write access to the table. This parameter is not supported when DB2REORG_OBJ_TABLE_INPLACE is specified as the **reorgType**.

DB2REORG_ALLOW_WRITE
Allow read and write access to the table. This parameter is not supported when DB2REORG_OBJ_TABLE_OFFLINE is specified as the **reorgType**.

DB2REORG_ALLOW_READ
Allow only read access to the table.

DB2REORG_CLEANUP_NONE
No clean up is required, used when DB2REORG_OBJ_INDEXESALL or DB2REORG_OBJ_INDEX are specified as the **reorgType**.

DB2REORG_CLEANUP_ALL
Clean up the committed pseudo deleted keys and committed pseudo empty pages, used when DB2REORG_OBJ_INDEXESALL or DB2REORG_OBJ_INDEX are specified as the **reorgType**.

DB2REORG_CLEANUP_PAGES
Clean up committed pseudo empty pages only, but do not clean up pseudo deleted keys on pages that are not pseudo empty, used when DB2REORG_OBJ_INDEXESALL or DB2REORG_OBJ_INDEX are specified as the **reorgType**.

DB2REORG_CONVERT_NONE
No conversion is required, used when DB2REORG_OBJ_INDEXESALL or DB2REORG_OBJ_INDEX are specified as the **reorgType**.

DB2REORG_CONVERT
Convert to type 2 index, used when DB2REORG_OBJ_INDEXESALL is specified as the **reorgType**. Type-1 indexes are no longer supported and are converted to type-2 indexes when the table is next accessed. As a result, this value is deprecated and might be removed in a future release.

DB2REORG_RESET_DICTIONARY
If the DB2REORG_LONGLOB option is also specified, DB2REORG_RESETDICTIONARY applies to the XML storage object of the table also. If the COMPRESS attribute for the table is YES then a new compression dictionary is built. All the rows processed during reorganization are subject to compression using this new dictionary. This dictionary replaces any previous dictionary in the object. If the COMPRESS attribute for the table is NO and the table object or the XML storage object does have an existing compression dictionary then reorg processing will remove the dictionary and all rows in the newly reorganized table will be in non-compressed format. This parameter is only supported for the DB2REORG_OBJ_TABLE_OFFLINE **reorgType**.

DB2REORG_KEEP_DICTIONARY
If DB2REORG_LONGLOB keyword is also specified, DB2REORG_KEEPDICTIONARY applies to the table object and the XML storage object of the table. If DB2REORG_LONGLOB is not specified, the following applies only to the table object.

If the COMPRESS attribute for the table is YES and a dictionary exists, it is kept. If the COMPRESS attribute for the table is YES and a dictionary does not exist, one is built, as the option defaults to DB2REORG_RESET_DICTIONARY in that case. All rows processed by reorganization are subject to compression. If the COMPRESS attribute for the table is NO, the dictionary will be retained (if one existed), and all rows in the newly reorganized table will be in non-compressed format. This parameter is only supported for the DB2REORG_OBJ_TABLE_OFFLINE **reorgType**.

nodeListFlag
Input. Specifies which nodes to reorganize. Valid values (defined in `db2ApiDf` header file, located in the `include` directory) are:

DB2REORG_NODE_LIST
Submit to all nodes in the nodelist array.

DB2REORG_ALL_NODES
Submit to all nodes in the database partition group.

DB2REORG_ALL_EXCEPT
Submit to all nodes except the ones specified by the nodelist parameter.

numNodes
Input. Number of nodes in the nodelist array.

pNodeList
A pointer to the array of node numbers.

reorgObject
Input. Specifies the type of object to be reorganized.

db2ReorgObject union parameters

tableStruct
Specifies the options for a table reorganization.

indexesAllStruct
Specifies the options for an index reorganization.

db2ReorgTable data structure parameters

pTableName
Input. Specifies the name of the table to reorganize.

pOrderByIndex
Input. Specifies the index to order the table by.

pSysTempSpace
Input. Specifies the system temporary table space where temporary objects are created. The REORG command may expand rows in cases where a column is added to a table (i.e. from ALTER TABLE ADD COLUMN) and the rows were inserted before the column was added. For a nonpartitioned table, this parameter must specify a table space with enough room to create the new table object. A partitioned table is reorganized a single data partition at a time. In this case, there must be enough free space in the table space to hold the largest data partition of the table.

If this parameter is not specified for a nonpartitioned table the table space the table resides in is used. If this parameter is not specified for a partitioned table, the table space where each data partition is located is used for temporary storage of that data partition. There must be enough free space in each data partition's table space to hold a copy of the data partition.

pLongTempSpace
Input. Specifies the temporary table space to create long objects (LONG VARCHAR and LOB columns) in during table reorganization. If the **pSysTempSpace** parameter is not specified, this parameter is ignored. If this parameter is not specified, but the **pSysTempSpace** parameter is specified, then DB2 will create the long data objects in the table space specified by the **pSysTempSpace** parameter, unless the page sizes differ.

When page sizes differ, if **pSysTempSpace** is specified, but this parameter is not, DB2 will attempt to find an existing table space with a matching page size to create the long objects in.

pPartitionName
Input. Specifies the name of the data partition to reorganize. Only supported for the MDC RECLAIM option (SQL2222N).

db2ReorgIndexesAll data structure parameters

pTableName
Input. Specifies the name of the table for index reorganization. If DB2REORG_OBJ_INDEX is specified as the **reorgType**, the **pTableName** parameter is not required and can be NULL. However, if the **pTableName** parameter is specified, it must be the table on which the index is defined.

pIndexName
Input. Specifies the name of the index to reorganize. This parameter is used only when the **reorgType** parameter is set to a value of DB2REORG_OBJ_INDEX otherwise set **pIndexName** parameter to NULL.

pPartitionName
Input. Specifies the name of the data partition whose indexes are to be reorganized. Only supported for the MDC RECLAIM option (SQL2222N).

db2gReorgTable data structure specific parameters

tableNameLen
Input. Specifies the length in bytes of **pTableName**.

orderByIndexLen
Input. Specifies the length in byte of **pOrderByIndex**.

sysTempSpaceLen
Input. Specifies the length in bytes of **pSysTempSpace**.

longTempSpaceLen
Input. Specifies the length of the name stored in the **pLongTempSpace**

partitionNameLen
Input. Specifies the length, in bytes, of **pPartitionName**. Only supported for the MDC RECLAIM option (SQL2222N).

pPartitionName
Input. Specifies the name of the data partition to reorganize. Only supported for the MDC RECLAIM option (SQL2222N).

db2gReorgIndexesAll data structure specific parameters

tableNameLen
Input. Specifies the length in bytes of **pTableName**.

indexNameLen
Input. Specifies the length in bytes of the **pIndexName** parameter.

partitionNameLen
Input. Specifies the length, in bytes, of **pPartitionName**. Only supported for the MDC RECLAIM option (SQL2222N).

pPartitionName
Input. Specifies the name of the data partition for the index. Only supported for the MDC RECLAIM option (SQL2222N).

Usage notes

- Performance of table access, index scans, and the effectiveness of index page prefetching can be adversely affected when the table data has been modified many times, becoming fragmented and unclustered. Use REORGCHK to determine whether a table or its indexes are candidates for reorganizing. All work will be committed and all open cursors will be closed during reorg processing. After reorganizing a table or its indexes, use db2Runstats to update the statistics and sqlarbnd to rebind the packages that use this table.
- If the table data is distributed onto several nodes and the reorganization fails on any of the affected nodes, then only the failing nodes will have the reorganization rolled back. If table reorganization is not successful, temporary files should not be deleted. The database manager uses these files to recover the database.
- For table reorganization, if the name of an index is specified, the database manager reorganizes the data according to the order in the index. To maximize performance, specify an index that is often used in SQL queries. If the name of an index is not specified, and if a clustering index exists, the data will be ordered according to the clustering index.
- The PCTFREE value of a table determines the amount of free space designated per page. If the value has not been set, the utility will fill up as much space as possible on each page.

- To complete a table space rollforward recovery following a table reorganization, both data and LONG table spaces must be rollforward enabled.
- If the table contains LOB columns not defined with the COMPACT option, the LOB DATA storage object can be significantly larger following table reorganization. This can be a result of the order in which the rows were reorganized, and the types of table spaces used (SMS/DMS).
- The following table illustrates the default table access chosen based on the type of reorg and table:

Table 171. Default table access chosen based on the type of reorg and table

Type of reorg and applicable flags which can affect the default table access		Access mode chosen for each table type	
reorgType	**reorgFlags (if applicable)**	**Non-partitioned table**	**Partitioned table**
DB2REORG_OBJ_TABLE_ OFFLINE		DB2REORG_ALLOW_ READ	DB2REORG_ALLOW_ NONE
DB2REORG_OBJ_TABLE_ OFFLINE		DB2REORG_ALLOW_ READ	DB2REORG_ALLOW_ READ (*)
DB2REORG_OBJ_TABLE_ INPLACE		DB2REORG_ALLOW_ WRITE	N/A
DB2REORG_OBJ_ INDEXESALL		DB2REORG_ALLOW_ READ	DB2REORG_ALLOW_ NONE
DB2REORG_OBJ_ INDEXESALL		N/A	DB2REORG_ALLOW_ READ
DB2REORG_OBJ_ INDEXESALL	DB2REORG_CLEANUP_ ALL, DB2REORG_CLEANUP_ PAGES	DB2REORG_ALLOW_ READ	DB2REORG_ALLOW_ READ
DB2REORG_OBJ_INDEX		N/A	DB2REORG_ALLOW_ READ
DB2REORG_OBJ_INDEX	DB2REORG_CLEANUP_ ALL, DB2REORG_CLEANUP_ PAGES	N/A	DB2REORG_ALLOW_ READ

(*) DB2REORG_ALLOW_READ if there are no nonpartitioned indexes; otherwise, DB2REORG_ALLOW_NONEN/A: Not applicable at this time since it is not supported.

Note: Some access modes may not be supported on certain types of tables or indexes. In these cases and where possible, the least restrictive access mode is used. (The most restrictive access mode being DB2REORG_ALLOW_NONE, followed by DB2REORG_ALLOW_READ, and then DB2REORG_ALLOW_WRITE, which is the least restrictive). As support for existing table or index types change, or new table or index types are provided, the default can change from a more restrictive access mode to a less restrictive mode. The least restrictive mode chosen for the default will not go beyond DB2REORG_ALLOW_READ when the **reorgType** is not DB2REORG_OBJ_TABLE_INPLACE. The default access mode is chosen when none of the DB2REORG_ALLOW_NONE, DB2REORG_ALLOW_READ, or DB2REORG_ALLOW_WRITE flags are specified.

- When reorganizing indexes, use the access option to allow other transactions either read-only or read-write access to the table.
- If an index reorganization with allow read or allow write access fails because the indexes need to be rebuilt, the reorganization will switch to allow no access and

then continue. A message will be written to both the administration notification log and the diagnostics log about the change in the access mode. When DB2REORG_OBJ_INDEX is specified for a partitioned table, indexes that need to be rebuilt are rebuilt offline, then the specified index is reorganized (assuming that it was not rebuilt). This reorganization uses the specified access mode (that is, the access mode will not change during processing). A message will be written to the administration notification log and the diagnostics log about the indexes being rebuilt offline.

- For non-inplace table reorganization, if neither DB2REORG_RESET_DICTIONARY or DB2REORG_KEEP_DICTIONARY is specified, the default is DB2REORG_KEEP_DICTIONARY.
- If an index reorganization with no access fails, some or all indexes will not be available and will be rebuilt on the next table access.
- This API cannot be used with:
  - Views or an index that is based on an index extension.
  - Declared temporary tables.
  - Created temporary tables.

db2Restore - Restore a database or table space

Recreates a damaged or corrupted database that has been backed up using the db2Backup API. The restored database is in the same state it was in when the backup copy was made.

This utility can also:

- Restore to a database with a name different from the database name in the backup image (in addition to being able to restore to a new database), the exception being a snapshot restore where the backup image database name must be the same.
- Restore DB2 databases that were created in the two previous releases.
- Restore from a table space level backup, or restore table spaces from within a database backup image.

Scope

This API only affects the database partition from which it is called.

Authorization

To restore to an existing database, one of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

To restore to a new database, one of the following:

- *sysadm*
- *sysctrl*

Required connection

Database, to restore to an existing database. This API automatically establishes a connection to the specified database and will release the connection when the restore operation finishes.

Instance and *database*, to restore to a new database. The instance attachment is required to create the database.

For snapshot restore, *instance* and *database* connections are required.

To restore to a new database at an instance different from the current instance (as defined by the value of the **DB2INSTANCE** environment variable), it is necessary to first attach to the instance where the new database will reside.

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Restore (
       db2Uint32 versionNumber,
       void * pDB2RestoreStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RestoreStruct
{
  char *piSourceDBAlias;
  char *piTargetDBAlias;
  char oApplicationId[SQLU_APPLID_LEN+1];
  char *piTimestamp;
  char *piTargetDBPath;
  char *piReportFile;
  struct db2TablespaceStruct *piTablespaceList;
  struct db2MediaListStruct *piMediaList;
  char *piUsername;
  char *piPassword;
  char *piNewLogPath;
  void *piVendorOptions;
  db2Uint32 iVendorOptionsSize;
  db2Uint32 iParallelism;
  db2Uint32 iBufferSize;
  db2Uint32 iNumBuffers;
  db2Uint32 iCallerAction;
  db2Uint32 iOptions;
  char *piComprLibrary;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
  char *piLogTarget;
  struct db2StoragePathsStruct *piStoragePaths;
  char *piRedirectScript;
} db2RestoreStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                        **tablespaces;
  db2Uint32 numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                        **locations;
  db2Uint32 numLocations;
  char locationType;
} db2MediaListStruct;

typedef SQL_STRUCTURE db2StoragePathsStruct
{
  char                        **storagePaths;
  db2Uint32 numStoragePaths;
```

```
} db2StoragePathsStruct;

SQL_API_RC SQL_API_FN
  db2gRestore (
       db2Uint32 versionNumber,
       void * pDB2gRestoreStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRestoreStruct
{
  char *piSourceDBAlias;
  db2Uint32 iSourceDBAliasLen;
  char *piTargetDBAlias;
  db2Uint32 iTargetDBAliasLen;
  char *poApplicationId;
  db2Uint32 iApplicationIdLen;
  char *piTimestamp;
  db2Uint32 iTimestampLen;
  char *piTargetDBPath;
  db2Uint32 iTargetDBPathLen;
  char *piReportFile;
  db2Uint32 iReportFileLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct *piMediaList;
  char *piUsername;
  db2Uint32 iUsernameLen;
  char *piPassword;
  db2Uint32 iPasswordLen;
  char *piNewLogPath;
  db2Uint32 iNewLogPathLen;
  void *piVendorOptions;
  db2Uint32 iVendorOptionsSize;
  db2Uint32 iParallelism;
  db2Uint32 iBufferSize;
  db2Uint32 iNumBuffers;
  db2Uint32 iCallerAction;
  db2Uint32 iOptions;
  char *piComprLibrary;
  db2Uint32 iComprLibraryLen;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
  char *piLogTarget;
  db2Uint32 iLogTargetLen;
  struct db2gStoragePathsStruct *piStoragePaths;
  char *piRedirectScript;
  db2Uint32 iRedirectScriptLen;
} db2gRestoreStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char *tablespaces;
  db2Uint32 numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char *locations;
  db2Uint32 numLocations;
  char locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2gStoragePathsStruct
{
  struct db2Char *storagePaths;
  db2Uint32 numStoragePaths;
} db2gStoragePathsStruct;
```

```
typedef SQL_STRUCTURE db2Char
{
   char *pioData;
   db2Uint32 iLength;
   db2Uint32 oLength;
} db2Char;
```

db2Restore API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter **pDB2RestoreStruct**.

pDB2RestoreStruct
Input. A pointer to the db2RestoreStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2RestoreStruct data structure parameters

piSourceDBAlias
Input. A string containing the database alias of the source database backup image.

piTargetDBAlias
Input. A string containing the target database alias. If this parameter is null, the value of the **piSourceDBAlias** parameter will be used.

oApplicationId
Output. The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

piTimestamp
Input. A string representing the time stamp of the backup image. This field is optional if there is only one backup image in the source specified.

piTargetDBPath
Input. A string containing the relative or fully qualified name of the target database directory on the server. Used if a new database is to be created for the restored backup; otherwise not used.

piReportFile
Input. The file name, if specified, must be fully qualified.

Note: This parameter is obsolete, but still defined.

piTablespaceList
Input. List of table spaces to be restored. Used when restoring a subset of table spaces from a database or table space backup image. For rebuild cases, this can be an include list or exclude list of table spaces used to rebuild your database. See the DB2TablespaceStruct structure. The following restrictions apply:

- The database must be recoverable (for non-rebuild cases only); that is, log retain or user exits must be enabled.
- The database being restored to must be the same database that was used to create the backup image. That is, table spaces can not be added to a database through the table space restore function.
- The rollforward utility will ensure that table spaces restored in a partitioned database environment are synchronized with any other

database partition containing the same table spaces. If a table space restore operation is requested and the **piTablespaceList** is NULL, the restore utility will attempt to restore all of the table spaces in the backup image.

- When restoring a table space that has been renamed since it was backed up, the new table space name must be used in the restore command. If the old table space name is used, it will not be found.
- In the case of rebuild, the list must be given for 3 of the 5 rebuild types: DB2RESTORE_ALL_TBSP_IN_DB_EXC, DB2RESTORE_ALL_TBSP_IN_IMG_EXC and DB2RESTORE_ALL_TBSP_IN_LIST.

piMediaList
Input. Source media for the backup image.

For more information, see the db2MediaListStruct structure below.

piUsername
Input. A string containing the user name to be used when attempting a connection. Can be NULL.

piPassword
Input. A string containing the password to be used with the user name. Can be NULL.

piNewLogPath
Input. A string representing the path to be used for logging after the restore has completed. If this field is null the default log path will be used.

piVendorOptions
Input. Used to pass information from the application to the vendor functions. This data structure must be flat; that is, no level of indirection is supported. Note that byte-reversal is not done, and the code page is not checked for this data.

iVendorOptionsSize
Input. The length in bytes of the **piVendorOptions** parameter, which cannot exceed 65535 bytes.

iParallelism
Input. Degree of parallelism (number of buffer manipulators). Minimum is 1. Maximum is 1024.

iBufferSize
Input. Backup buffer size in 4 KB allocation units (pages). Minimum is 8 units. The size entered for a restore must be equal to or an integer multiple of the buffer size used to produce the backup image.

iNumBuffers
Input. Specifies number of restore buffers to be used.

iCallerAction
Input. Specifies action to be taken. Valid values (defined in `db2ApiDf` header file, located in the include directory) are:

- DB2RESTORE_RESTORE - Start the restore operation.
- DB2RESTORE_NOINTERRUPT - Start the restore. Specifies that the restore will run unattended, and that scenarios which normally require user intervention will either be attempted without first returning to the

caller, or will generate an error. Use this caller action, for example, if it is known that all of the media required for the restore have been mounted, and utility prompts are not desired.

- DB2RESTORE_CONTINUE - Continue the restore after the user has performed some action requested by the utility (mount a new tape, for example).
- DB2RESTORE_TERMINATE - Terminate the restore after the user has failed to perform some action requested by the utility.
- DB2RESTORE_DEVICE_TERMINATE - Remove a particular device from the list of devices used by restore. When a particular device has exhausted its input, restore will return a warning to the caller. Call restore again with this caller action to remove the device which generated the warning from the list of devices being used.
- DB2RESTORE_PARM_CHK - Used to validate parameters without performing a restore. This option does not terminate the database connection after the call returns. After a successful return of this call, it is expected that the user will issue another call to this API with the **iCallerAction** parameter set to the value DB2RESTORE_CONTINUE to continue with the restore.
- DB2RESTORE_PARM_CHK_ONLY - Used to validate parameters without performing a restore. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.
- DB2RESTORE_TERMINATE_INCRE - Terminate an incremental restore operation before completion.
- DB2RESTORE_RESTORE_STORDEF - Initial call. Table space container redefinition requested.
- DB2RESTORE_STORDEF_NOINTERRUPT - Initial call. The restore will run uninterrupted. Table space container redefinition requested.

iOptions

Input. A bitmap of restore properties. The options are to be combined using the bitwise OR operator to produce a value for **iOptions**. Valid values (defined in `db2ApiDf` header file, located in the include directory) are:

- DB2RESTORE_OFFLINE - Perform an offline restore operation.
- DB2RESTORE_ONLINE - Perform an online restore operation.
- DB2RESTORE_DB - Restore all table spaces in the database. This must be run offline.
- DB2RESTORE_TABLESPACE - Restore only the table spaces listed in the **piTablespaceList** parameter from the backup image. This can be online or offline.
- DB2RESTORE_HISTORY - Restore only the history file.
- DB2RESTORE_COMPR_LIB - Indicates that the compression library is to be restored. This option cannot be used simultaneously with any other type of restore process. If the object exists in the backup image, it will be restored into the database directory. If the object does not exist in the backup image, the restore operation will fail.
- DB2RESTORE_LOGS - Specifies that only the set of log files contained in the backup image are to be restored. If the backup image does not include log files, the restore operation will fail. If this option is specified, the **piLogTarget** parameter must also be specified.

- DB2RESTORE_INCREMENTAL - Perform a manual cumulative restore operation.
- DB2RESTORE_AUTOMATIC - Perform an automatic cumulative (incremental) restore operation. Must be specified with DB2RESTORE_INCREMENTAL.
- DB2RESTORE_ROLLFWD - Place the database in rollforward pending state after it has been successfully restored.
- DB2RESTORE_NOROLLFWD - Do not place the database in rollforward pending state after it has been successfully restored. This cannot be specified for backups taken online or for table space level restores. If, following a successful restore, the database is in roll-forward pending state, the db2Rollforward API must be called before the database can be used.
- DB2RESTORE_GENERATE_SCRIPT - Create a script, that can be used to perform a redirected restore. **piRedirectScript** must contain a valid file name. The **iCallerAction** need to be either DB2RESTORE_RESTORE_STORDEF or DB2RESTORE_STORDEF_NOINTERRUPT.

The following values should be used for rebuild operations only:

- DB2RESTORE_ALL_TBSP_IN_DB - Restores the database with all the table spaces known to the database at the time of the image being restored. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_DB_EXC - Restores the database with all the table spaces known to the database at the time of the image being restored except for those specified in the list pointed to by the **piTablespaceList** parameter. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_IMG - Restores the database with only the table spaces in the image being restored. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_IMG_EXC - Restores the database with only the table spaces in the image being restored except for those specified in the list pointed to by the **piTablespaceList** parameter. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_LIST - Restores the database with only the table spaces specified in the list pointed to by the **piTablespaceList** parameter. This rebuild overwrites a database if it already exists.

NOTE: If the backup image is of a recoverable database, then WITHOUT ROLLING FORWARD (DB2RESTORE_NOROLLFWD) cannot be specified with any of the above rebuild actions.

piComprLibrary

Input. Indicates the name of the external library to use to decompress the backup image if the image is compressed. The name must be a fully-qualified path that refers to a file on the server. If the value is a null pointer or a pointer to an empty string, the DB2 database system attempts to use the library stored in the image. If the backup is not compressed, the value of this parameter will be ignored. If the specified library is not found, the restore operation will fail.

piComprOptions

Input. This API parameter describes a block of binary data that will be passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so

any issues of byte-reversal or code-page conversion must be handled by the compression library. If the first character of the data block is '@', the remainder of the data is interpreted as the name of a file residing on the server. The DB2 database system then replaces the contents of the **piComprOptions** and **iComprOptionsSize** parameters with the contents and size of this file and passes these new values to the initialization routine.

iComprOptionsSize
Input. A four-byte unsigned integer that represents the size of the block of data passed as **piComprOptions**. The **iComprOptionsSize** parameter should be zero if and only if the **piComprOptions** value is a null pointer.

piLogTarget
Input. Specifies the absolute path of a directory on the database server that must be used as the target directory for extracting log files from a backup image. If this parameter is specified, any log files included in the backup image are extracted into the target directory. If this parameter is not specified, log files included in the backup image are not extracted. To extract only the log files from the backup image, DB2RESTORE_LOGS value should be passed to the **iOptions** parameter.

For snapshot restore, one of the following must be given:

- DB2RESTORE_LOGTARGET_INCLUDE "INCLUDE"

  Restore log directory volumes from the snapshot image. If this option is specified and the backup image contains log directories, then they will be restored. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If existing log directories on disk conflict with the log directories in the backup image, then an error will be returned.
- DB2RESTORE_LOGTARGET_EXCLUDE "EXCLUDE"

  Do not restore log directory volumes. If this option is specified, then log directories will not be restored from the backup image. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If a path belonging to the database is restored and a log directory will implicitly be restored because of this, thus causing a log directory to be overwritten, an error will be returned.
- DB2RESTORE_LOGTARGET_INCFORCE "INCLUDE FORCE"

  Allow existing log directories to be overwritten and replaced when restoring the snapshot image. If this option is specified and the backup image contains log directories, then they will be restored. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If existing log directories on disk conflict with the log directories in the backup image, then they will be overwritten by those from the backup image.
- DB2RESTORE_LOGTARGET_EXCFORCE "EXCLUDE FORCE"

  Allow existing log directories to be overwritten and replaced when restoring the snapshot image. If this option is specified, then log directories will not be restored from the backup image. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If a path belonging to the database is restored and a log directory will implicitly be restored because of this, thus causing a log directory to be overwritten, the restore will go ahead and overwrite the conflicting log directory.

where DB2RESTORE_LOGTARGET_EXCLUDE is the default.

piStoragePaths
Input. A structure containing fields that describe a list of storage paths used for automatic storage. Set this to NULL if automatic storage is not enabled for the database.

piRedirectScript
Input. The file name for the redirect restore script that will be created on client side. The file name can be specified relative or absolute. The **iOptions** field need to have the DB2RESTORE_GENERATE_SCRIPT bit set.

db2TablespaceStruct data structure specific parameters

tablespaces
Input. A pointer to the list of table spaces to be backed up. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numTablespaces
Input. Number of entries in the **tablespaces** parameter.

db2MediaListStruct data structure parameters

locations
Input. A pointer to the list of media locations. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numLocations
Input. The number of entries in the **locations** parameter.

locationType
Input. A character indicating the media type. Valid values (defined in `sqlutil` header file, located in the include directory) are:

SQLU_LOCAL_MEDIA: 'L'
Local devices (tapes, disks, diskettes, or named pipes).

SQLU_XBSA_MEDIA: 'X'
XBSA interface.

SQLU_TSM_MEDIA: 'A'
Tivoli Storage Manager.

SQLU_OTHER_MEDIA: 'O'
Vendor library.

SQLU_SNAPSHOT_MEDIA: 'F'
Specifies that the data is to be restored from a snapshot backup.

You cannot use SQLU_SNAPSHOT_MEDIA with any of the following:

- caller actions: DB2RESTORE_RESTORE_STORDEF, DB2RESTORE_STORDEF_NOINTERRUPT, DB2RESTORE_TERMINATE_INCRE
- DB2RESTORE_REPLACE_HISTORY
- DB2RESTORE_TABLESPACE
- DB2RESTORE_COMPR_LIB
- DB2RESTORE_INCREMENTAL
- DB2RESTORE_HISTORY

- DB2RESTORE_LOGS
- **piStoragePaths** - it must be NULL or empty in order to use it
- **piTargetDBPath**
- **piTargetDBAlias**
- **piNewLogPath**
- **iNumBuffers**
- **iBufferSize**
- **piRedirectScript**
- **iRedirectScriptLen**
- **iParallelism**
- **piComprLibrary**, **iComprLibraryLen**, **piComprOptions**, or **iComprOptionsSize**
- **numLocations** field of this structure must be 1 for snapshot restore

Also, you cannot use the SNAPSHOT parameter with any restore operation that involves a table space list.

The default behavior when restoring data from a snapshot backup image will be a FULL DATABASE OFFLINE restore of all paths that make up the database including all containers, local volume directory, database path (**DBPATH**), primary log and mirror log paths of the most recent snapshot backup if no timestamp is provided (**INCLUDE LOGS** is the default for all snapshot backups unless **EXCLUDE LOGS** is explicitly stated). If a timestamp is provided then that snapshot backup image will be restored.

Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage SAN Volume Controller
- IBM Enterprise Storage Server Model 800
- IBM System Storage DS6000
- IBM System Storage DS8000
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS

db2StoragePathsStruct data structure parameters

storagePaths
Input. An array of strings containing fully qualified names of storage paths on the server that will be used for automatic storage table spaces. In a multi-partition database the same storage paths are used on all database partitions. If a multi-partition database is being restored with new storage paths, then the catalog partition must be restored before any other database partitions are restored.

numStoragePaths
Input. The number of storage paths in the **storagePaths** parameter of the db2StoragePathsStruct structure.

db2gRestoreStruct data structure specific parameters

iSourceDBAliasLen
Input. Specifies the length in bytes of the **piSourceDBAlias** parameter.

iTargetDBAliasLen
Input. Specifies the length in bytes of the **piTargetDBAlias** parameter.

iApplicationIdLen
Input. Specifies the length in bytes of the **poApplicationId** parameter. Should be equal to SQLU_APPLID_LEN + 1. The constant SQLU_APPLID_LEN is defined in `sqlutil` header file that is located in the include directory.

iTimestampLen
Input. Specifies the length in bytes of the **piTimestamp** parameter.

iTargetDBPathLen
Input. Specifies the length in bytes of the **piTargetDBPath** parameter.

iReportFileLen
Input. Specifies the length in bytes of the **piReportFile** parameter.

iUsernameLen
Input. Specifies the length in bytes of the **piUsername** parameter. Set to zero if no user name is provided.

iPasswordLen
Input. Specifies the length in bytes of the **piPassword** parameter. Set to zero if no password is provided.

iNewLogPathLen
Input. Specifies the length in bytes of the **piNewLogPath** parameter.

iLogTargetLen
Input. Specifies the length in bytes of the **piLogTarget** parameter.

iRedirectScriptLen
Input. A four-byte unsigned integer representing the length in bytes of the name of the library specified in **piRedirectScript**. Set to zero if no script name is given.

db2Char data structure parameters

pioData
A pointer to a character data buffer. If NULL, no data will be returned.

iLength
Input. The size of the **pioData** buffer.

oLength
Output. The number of valid characters of data in the **pioData** buffer.

Usage notes

- For offline restore, this utility connects to the database in exclusive mode. The utility fails if any application, including the calling application, is already connected to the database that is being restored. In addition, the request will fail if the restore utility is being used to perform the restore, and any application, including the calling application, is already connected to any database on the same workstation. If the connect is successful, the API locks out other applications until the restore is completed.

- The current database configuration file will not be replaced by the backup copy unless it is unusable. In this case, if the file is replaced, a warning message is returned.
- The database or table space must have been backed up using the db2Backup API.
- If the caller action value is DB2RESTORE_NOINTERRUPT, the restore continues without prompting the application. If the caller action value is DB2RESTORE_RESTORE, and the utility is restoring to an existing database, the utility returns control to the application with a message requesting some user interaction. After handling the user interaction, the application calls RESTORE DATABASE again, with the caller action value set to indicate whether processing is to continue (DB2RESTORE_CONTINUE) or terminate (DB2RESTORE_TERMINATE) on the subsequent call. The utility finishes processing, and returns an SQLCODE in the sqlca.
- To close a device when finished, set the caller action value to DB2RESTORE_DEVICE_TERMINATE. If, for example, a user is restoring from 3 tape volumes using 2 tape devices, and one of the tapes has been restored, the application obtains control from the API with an SQLCODE indicating end of tape. The application can prompt the user to mount another tape, and if the user indicates ″no more″, return to the API with caller action value DB2RESTORE_DEVICE_TERMINATE to signal end of the media device. The device driver will be terminated, but the rest of the devices involved in the restore will continue to have their input processed until all segments of the restore set have been restored (the number of segments in the restore set is placed on the last media device during the backup process). This caller action can be used with devices other than tape (vendor supported devices).
- To perform a parameter check before returning to the application, set caller action value to DB2RESTORE_PARM_CHK.
- Set caller action value to DB2RESTORE_RESTORE_STORDEF when performing a redirected restore; used in conjunction with the sqlbstsc API.
- If a system failure occurs during a critical stage of restoring a database, the user will not be able to successfully connect to the database until a successful restore is performed. This condition will be detected when the connection is attempted, and an error message is returned. If the backed-up database is not configured for roll-forward recovery, and there is a usable current configuration file with either of these parameters enabled, following the restore, the user will be required to either take a new backup of the database, or disable the log retain and user exit parameters before connecting to the database.
- Although the restored database will not be dropped (unless restoring to a nonexistent database), if the restore fails, it will not be usable.
- If the restore type specifies that the history file in the backup is to be restored, it will be restored over the existing history file for the database, effectively erasing any changes made to the history file after the backup that is being restored. If this is undesirable, restore the history file to a new or test database so that its contents can be viewed without destroying any updates that have taken place.
- If, at the time of the backup operation, the database was enabled for roll forward recovery, the database can be brought to the state it was in prior to the occurrence of the damage or corruption by issuing db2Rollforward after successful execution of db2Restore. If the database is recoverable, it will default to roll forward pending state after the completion of the restore.
- If the database backup image is taken offline, and the caller does not want to roll forward the database after the restore, the DB2RESTORE_NOROLLFWD option can be used for the restore. This results in the database being usable

immediately after the restore. If the backup image is taken online, the caller must roll forward through the corresponding log records at the completion of the restore.

- To restore log files from a backup image that contains them, the **LOGTARGET** option must be specified, assuming a fully qualified and valid path exists on the DB2 server. If those conditions are satisfied, the restore utility writes the log files from the image to the target path. If **LOGTARGET** is specified during a restoration of a backup image that does not include logs, the restore operation returns an error before attempting to restore any table space data. A restore operation also fails with an error if an invalid or read-only **LOGTARGET** path is specified.
- If any log files exist in the **LOGTARGET** path at the time the RESTORE DATABASE command is issued, a warning prompt is returned to user. This warning is not returned if **WITHOUT PROMPTING** is specified.
- During a restore operation in which a **LOGTARGET** is specified, if any log file cannot be extracted, the restore operation fails and returns an error. If any of the log files being extracted from the backup image have the same name as an existing file in the **LOGTARGET** path, the restore operation fails and an error is returned. The restore utility does not overwrite existing log files in the **LOGTARGET** directory.
- You can restore only the saved log set from a backup image. To indicate that only the log files are to be restored, specify the LOGS option in addition to the **LOGTARGET** path. Specifying the **LOGS** option without a **LOGTARGET** path results in an error. If any problem occurs while restoring log files in this mode the restore operation terminates immediately and an error is returned.
- During an automatic incremental restore operation, only the logs included in the target image of the restore operation are retrieved from the backup image. Any logs that are included in intermediate images that are referenced during the incremental restore process are not extracted from those intermediate backup images. During a manual incremental restore operation, the **LOGTARGET** path should be specified only with the final restore command.
- If a backup is compressed, the DB2 database system detects this state and automatically decompresses the data before restoring it. If a library is specified on the db2Restore API, it is used for decompressing the data. If a library is not specified on the db2Restore API, the library stored in the backup image is used. And if there is no library stored in the backup image, the data cannot be decompressed and the restore operation fails.
- If the compression library is being restored from a backup image (either explicitly by specifying the DB2RESTORE_COMPR_LIB restore type or implicitly by performing a normal restoration of a compressed backup), the restore operation must be done on the same platform and operating system that the backup was taken on. If the platforms are different, the restore operation will fail, even when the DB2 database system normally supports cross-platform restore operations involving the two systems.
- If restoring a database that is enabled for automatic storage, the storage paths associated with the database can be redefined or they can remain as they were previously. To keep the storage path definitions as is, do not provide any storage paths as part of the restore operation. Otherwise, specify a new set of storage paths to associate with the database. Automatic storage table spaces will be automatically redirected to the new storage paths during the restore operation.

Snapshot restore

Like a traditional (non-snapshot) restore, the default behavior when restoring a snapshot backup image will be to NOT restore the log directories — DB2RESTORE_LOGTARGET_EXCLUDE.

If the DB2 manager detects that any log directory's group ID is shared among any of the other paths to be restored, then an error is returned. In this case, DB2RESTORE_LOGTARGET_INCLUDE or DB2RESTORE_LOGTARGET_INCFORCE must be specified, as the log directories must be part of the restore.

The DB2 manager will make all efforts to save existing log directories (primary, mirror and overflow) before the restore of the paths from the backup image takes place.

If you wish the log directories to be restored and the DB2 manager detects that the preexisting log directories on disk conflict with the log directories in the backup image, then the DB2 manager will report an error. In such a case, if you have specified DB2RESTORE_LOGTARGET_INCFORCE, then this error will be suppressed and the log directories from the image will be restored, deleting whatever existed beforehand.

There is a special case in which the DB2RESTORE_LOGTARGET_EXCLUDE option is specified and a log directory path resides under the database directory (for example, /NODE*xxxx*/SQL*xxxxx*/SQLOGDIR/). In this case, a restore would still overwrite the log directory as the database path, and all of the contents beneath it, would be restored. If the DB2 manager detects this scenario and log files exist in this log directory, then an error will be reported. If you specify DB2RESTORE_LOGTARGET_EXCLUDE, then this error will be suppressed and those log directories from the backup image will overwrite the conflicting log directories on disk.

db2Rollforward - Roll forward a database

Recovers a database by applying transactions recorded in the database log files. Called after a database or a table space backup has been restored, or if any table spaces have been taken offline by the database due to a media error. The database must be recoverable (that is, either the **logarchmeth1** database configuration parameter or the **logarchmeth2** database configuration parameter must be set to a value other than OFF) before the database can be recovered with rollforward recovery.

Scope

In a partitioned database environment, you must call this API from the catalog partition. The partitions that are rolled forward depend on what you specify in the TO clause:

- A point-in-time rollforward call affects all database partition servers that are listed in the db2nodes.cfg file.
- An END OF LOGS rollforward call affects the database partition servers that are specified in the ON DATABASE PARTITION clause. If no database partition servers are specified, the rollforward call affects all database partition servers that are listed in the db2nodes.cfg file.
- A database or table space rollforward call specifying end of backup affects all database partitions servers that are listed in the db2nodes.cfg file.

If all of the transactions on a particular database partition server have already been applied to the current database, and therefore none of those transactions need to be rolled forward, that database partition server is ignored.

When you roll forward a partitioned table to a certain point in time, you must also roll forward the table spaces that contain that table to the same point in time. However, when you roll forward a table space, you do not have to roll forward all the tables in that table space.

Authorization

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection

None. This API establishes a database connection.

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Rollforward (
       db2Uint32 versionNumber,
       void * pDB2RollforwardStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RollforwardStruct
{
  struct db2RfwdInputStruct *piRfwdInput;
  struct db2RfwdOutputStruct *poRfwdOutput;
} db2RollforwardStruct;

typedef SQL_STRUCTURE db2RfwdInputStruct
{
  sqluint32 iVersion;
  char *piDbAlias;
  db2Uint32 iCallerAction;
  char *piStopTime;
  char *piUserName;
  char *piPassword;
  char *piOverflowLogPath;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2Uint32 iConnectMode;
  struct sqlu_tablespace_bkrst_list *piTablespaceList;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piDroppedTblID;
  char *piExportDir;
  db2Uint32 iRollforwardFlags;
} db2RfwdInputStruct;

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char *poApplicationId;
```

```
  sqlint32 *poNumReplies;
  struct sqlurf_info *poNodeInfo;
  db2Uint32 oRollforwardFlags;
} db2RfwdOutputStruct;

SQL_STRUCTURE sqlurf_newlogpath
{
   SQL_PDB_NODE_TYPE nodenum;
   unsigned short  pathlen;
   char logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
};

typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
   sqlint32 num_entry;
   struct sqlu_tablespace_entry *tablespace;
} sqlu_tablespace_bkrst_list;

typedef SQL_STRUCTURE sqlu_tablespace_entry
{
   sqluint32 reserve_len;
   char tablespace_entry[SQLU_MAX_TBS_NAME_LEN+1];
   char filler[1];
} sqlu_tablespace_entry;

SQL_STRUCTURE sqlurf_info
{
   SQL_PDB_NODE_TYPE nodenum;
   sqlint32 state;
   unsigned char   nextarclog[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   firstarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   lastarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   lastcommit[SQLUM_TIMESTAMP_LEN+1];
};

SQL_API_RC SQL_API_FN
  db2gRollforward (
       db2Uint32 versionNumber,
       void * pDB2gRollforwardStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRollforwardStruct
{
  struct db2gRfwdInputStruct *piRfwdInput;
  struct db2RfwdOutputStruct *poRfwdOutput;
} db2gRollforwardStruct;

typedef SQL_STRUCTURE db2gRfwdInputStruct
{
  db2Uint32 iDbAliasLen;
  db2Uint32 iStopTimeLen;
  db2Uint32 iUserNameLen;
  db2Uint32 iPasswordLen;
  db2Uint32 iOvrflwLogPathLen;
  db2Uint32 iDroppedTblIDLen;
  db2Uint32 iExportDirLen;
  sqluint32 iVersion;
  char *piDbAlias;
  db2Uint32 iCallerAction;
  char *piStopTime;
  char *piUserName;
  char *piPassword;
  char *piOverflowLogPath;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2Uint32 iConnectMode;
  struct sqlu_tablespace_bkrst_list *piTablespaceList;
```

```
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piDroppedTblID;
  char *piExportDir;
  db2Uint32 iRollforwardFlags;
} db2gRfwdInputStruct;
```

db2Rollforward API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter.

pDB2RollforwardStruct
Input. A pointer to the db2RollforwardStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2RollforwardStruct data structure parameters

piRfwdInput
Input. A pointer to the db2RfwdInputStruct structure.

poRfwdOutput
Output. A pointer to the db2RfwdOutputStruct structure.

db2RfwdInputStruct data structure parameters

iVersion
Input. The version ID of the rollforward parameters. It is defined as SQLUM_RFWD_VERSION.

piDbAlias
Input. A string containing the database alias. This is the alias that is cataloged in the system database directory.

iCallerAction
Input. Specifies action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_ROLLFWD
Rollforward to the point in time specified by the piStopTime parameter. For database rollforward, the database is left in rollforward-pending state. For table space rollforward to a point in time, the table spaces are left in rollforward-in-progress state.

DB2ROLLFORWARD_STOP
End roll-forward recovery by rolling forward the database using available log files and then rolling it back. Uncommitted transactions are backed out and the rollforward-pending state of the database or table spaces is turned off. A synonym for this value is DB2ROLLFORWARD_RFWD_COMPLETE.

DB2ROLLFORWARD_RFWD_STOP
Rollforward to the point in time specified by piStopTime, and end roll-forward recovery. The rollforward-pending state of the database or table spaces is turned off. A synonym for this value is DB2ROLLFORWARD_RFWD_COMPLETE.

DB2ROLLFORWARD_QUERY
Query values for nextarclog, firstarcdel, lastarcdel, and lastcommit. Return database status and a node number.

DB2ROLLFORWARD_PARM_CHECK
Validate parameters without performing the roll forward.

DB2ROLLFORWARD_CANCEL
Cancel the rollforward operation that is currently running. The database or table space are put in recovery pending state.

Note: This option cannot be used while the rollforward is actually running. It can be used if the rollforward is paused (that is, waiting for a STOP), or if a system failure occurred during the rollforward. It should be used with caution.
Rolling databases forward may require a load recovery using tape devices. The rollforward API will return with a warning message if user intervention on a device is required. The API can be called again with one of the following three caller actions:

DB2ROLLFORWARD_LOADREC_CONT
Continue using the device that generated the warning message (for example, when a new tape has been mounted).

DB2ROLLFORWARD_DEVICE_TERM
Stop using the device that generated the warning message (for example, when there are no more tapes).

DB2ROLLFORWARD_LOAD_REC_TERM
Terminate all devices being used by load recovery.

piStopTime
Input. A character string containing a time stamp in ISO format. Database recovery will stop when this time stamp is exceeded. Specify SQLUM_INFINITY_TIMESTAMP to roll forward as far as possible. May be NULL for DB2ROLLFORWARD_QUERY, DB2ROLLFORWARD_PARM_CHECK, and any of the load recovery (DB2ROLLFORWARD_LOADREC_xxx) caller actions.

piUserName
Input. A string containing the user name of the application. Can be NULL.

piPassword
Input. A string containing the password of the supplied user name (if any). Can be NULL.

piOverflowLogPath
Input. This parameter is used to specify an alternate log path to be used. In addition to the active log files, archived log files need to be moved (by the user) into the logpath before they can be used by this utility. This can be a problem if the database does not have sufficient space in the logpath. The overflow log path is provided for this reason. During roll-forward recovery, the required log files are searched, first in the logpath, and then in the overflow log path. The log files needed for table space roll-forward recovery can be brought into either the logpath or the overflow log path. If the caller does not specify an overflow log path, the default value is the logpath. In a partitioned database environment, the overflow log path must be a valid, fully qualified path; the default path is the default overflow log path for each node. In a single-partition database environment, the overflow log path can be relative if the server is local.

iNumChngLgOvrflw
Input. Partitioned database environments only. The number of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

piChngLogOvrflw
Input. Partitioned database environments only. A pointer to a structure containing the fully qualified names of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

iConnectMode
Input. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_OFFLINE
Offline roll forward. This value must be specified for database roll-forward recovery.

DB2ROLLFORWARD_ONLINE
Online roll forward.

piTablespaceList
Input. A pointer to a structure containing the names of the table spaces to be rolled forward to the end-of-logs or to a specific point in time. If not specified, the table spaces needing rollforward will be selected.

For partitioned tables, point in time (PIT) roll-forward of a table space containing any piece of a partitioned table must also roll forward all of the other table spaces in which that table resides to the same point in time. Roll forward to the end of the logs for a single table space containing a piece of a partitioned table is still allowed.

If a partitioned table has any attached, detached or dropped data partitions, then PIT roll-forward must include all table spaces for these data partitions as well. To determine if a partitioned table has any attached, detached, or dropped data partitions, query the Status field of the SYSDATAPARTITIONS catalog table.

Because a partitioned table can reside in multiple table spaces, it is generally necessary to roll forward multiple table spaces. Data that is recovered via dropped table recovery is written to the export directory specified in the piExportDir parameter. It is possible to roll forward all table spaces in one command, or do repeated roll-forward operations for subsets of the table spaces involved. A warning will be written to the notify log if the db2Rollforward API did not specify the full set of the table spaces necessary to recover all the data for the table. A warning will be returned to the user with full details of all partitions not recovered on the command found in the administration notification log.

Allowing the roll forward of a subset of the table spaces makes it easier to deal with cases where there is more data to be recovered than can fit into a single export directory.

iAllNodeFlag
Input. Partitioned database environments only. Indicates whether the rollforward operation is to be applied to all database partition servers defined in db2nodes.cfg. Valid values are:

DB2_NODE_LIST
Apply to database partition servers in a list that is passed in piNodeList.

DB2_ALL_NODES
Apply to all database partition servers. This is the default value. The piNodeList parameter must be set to NULL, if this value is used.

DB2_ALL_EXCEPT
Apply to all database partition servers except those in a list that is passed in piNodeList.

DB2_CAT_NODE_ONLY
Apply to the catalog partition only. The piNodeList parameter must be set to NULL, if this value is used.

iNumNodes
Input. Specifies the number of database partition servers in the piNodeList array.

piNodeList
Input. A pointer to an array of database partition server numbers on which to perform the roll-forward recovery.

iNumNodeInfo
Input. Defines the size of the output parameter poNodeInfo, which must be large enough to hold status information from each database partition that is being rolled forward. In a single-partition database environment, this parameter should be set to 1. The value of this parameter should be the same as the number of database partition servers for which this API is being called.

piDroppedTblID
Input. A string containing the ID of the dropped table whose recovery is being attempted. For partitioned tables, the drop-table-id identifies the table as a whole, so that all data partitions of the table can be recovered in a single roll-forward command.

piExportDir
Input. The name of the directory into which the dropped table data will be exported.

iRollforwardFlags
Input. Specifies the rollforward flags. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_EMPTY_FLAG
No flags specified.

DB2ROLLFORWARD_LOCAL_TIME
Allows the user to rollforward to a point in time that is the user's local time rather than GMT time. This makes it easier for users to rollforward to a specific point in time on their local machines, and eliminates potential user errors due to the translation of local to GMT time.

DB2ROLLFORWARD_NO_RETRIEVE
Controls which log files to be rolled forward on the standby machine by allowing the user to disable the retrieval of archived logs. By controlling the log files to be rolled forward, one can ensure that the standby machine is X hours behind the production

machine, to prevent the user affecting both systems. This option is useful if the standby system does not have access to archive, for example, if TSM is the archive, it only allows the original machine to retrieve the files. It will also remove the possibility that the standby system would retrieve an incomplete log file while the production system is archiving a file and the standby system is retrieving the same file.

DB2ROLLFORWARD_END_OF_BACKUP
Specifies that the database should be rolled forward to the *minimum recovery time*.

db2RfwdOutputStruct data structure parameters

poApplicationId
Output. The application ID.

poNumReplies
Output. The number of replies received.

poNodeInfo
Output. Database partition reply information.

oRollforwardFlags
Output. Rollforward output flags. Valid values are:

DB2ROLLFORWARD_OUT_LOCAL_TIME
Indicates to user that the last committed transaction timestamp is displayed in local time rather than UTC. Local time is based on the server's local time, not on the client's. In a partitioned database environment, local time is based on the catalog partition's local time.

sqlurf_newlogpath data structure parameters

nodenum
Input. The number of the database partition that this structure details.

pathlen
Input. The total length of the logpath field.

logpath
Input. A fully qualified path to be used for a specific node for the rollforward operation.

sqlu_tablespace_bkrst_list data structure parameters

num_entry
Input. The number of structures contained in the list pointed to by the table space parameter.

tablespace
Input. A pointer to a list of sqlu_tablespace_entry structures.

sqlu_tablespace_entry data structure parameters

reserve_len
Input. Specifies the length in bytes of the tablespace_entry parameter.

tablespace_entry
Input. The name of the table space to rollforward.

filler Filler used for proper alignment of data structure in memory.

sqlurf_info data structure parameters

nodenum
Output. The number of the database partition that this structure contains information for.

state Output. The current state of the database or table spaces that were included in the rollforward on a database partition.

nextarclog
Output. If the rollforward has completed, this field will be empty. If the rollforward has not yet completed, this will be the name of the next log file which will be processed for the rollforward.

firstarcdel
Output. The first log file replayed by the rollforward.

lastarcdel
Output. The last log file replayed by the rollforward.

lastcommit
Output. The time of the last committed transaction.

db2gRfwdInputStruct data structure specific parameters

iDbAliasLen
Input. Specifies the length in bytes of the database alias.

iStopTimeLen
Input. Specifies the length in bytes of the stop time parameter. Set to zero if no stop time is provided.

iUserNameLen
Input. Specifies the length in bytes of the user name. Set to zero if no user name is provided.

iPasswordLen
Input. Specifies the length in bytes of the password. Set to zero if no password is provided.

iOverflowLogPathLen
Input. Specifies the length in bytes of the overflow log path. Set to zero if no overflow log path is provided.

iDroppedTblIDLen
Input. Specifies the length in bytes of the dropped table ID (piDroppedTblID parameter). Set to zero if no dropped table ID is provided.

iExportDirLen
Input. Specifies the length in bytes of the dropped table export directory (piExportDir parameter). Set to zero if no dropped table export directory is provided.

Usage notes

The database manager uses the information stored in the archived and the active log files to reconstruct the transactions performed on the database since its last backup.

The action performed when this API is called depends on the rollforward_pending flag of the database prior to the call. This can be queried using db2CfgGet - Get

Configuration Parameters. The rollforward_pending flag is set to DATABASE if the database is in roll-forward pending state. It is set to TABLESPACE if one or more table spaces are in SQLB_ROLLFORWARD_PENDING or SQLB_ROLLFORWARD_IN_PROGRESS state. The rollforward_pending flag is set to NO if neither the database nor any of the table spaces needs to be rolled forward.

If the database is in roll-forward pending state when this API is called, the database will be rolled forward. Table spaces are returned to normal state after a successful database roll-forward, unless an abnormal state causes one or more table spaces to go offline. If the rollforward_pending flag is set to TABLESPACE, only those table spaces that are in roll-forward pending state, or those table spaces requested by name, will be rolled forward.

Note: If table space rollforward terminates abnormally, table spaces that were being rolled forward will be put in SQLB_ROLLFORWARD_IN_PROGRESS state. In the next invocation of ROLLFORWARD DATABASE, only those table spaces in SQLB_ROLLFORWARD_IN_PROGRESS state will be processed. If the set of selected table space names does not include all table spaces that are in SQLB_ROLLFORWARD_IN_PROGRESS state, the table spaces that are not required will be put into SQLB_RESTORE_PENDING state.

If the database is not in roll-forward pending state and no point in time is specified, any table spaces that are in rollforward-in-progress state will be rolled forward to the end of logs. If no table spaces are in rollforward-in-progress state, any table spaces that are in rollforward pending state will be rolled forward to the end of logs.

This API reads the log files, beginning with the log file that is matched with the backup image. The name of this log file can be determined by calling this API with a caller action of DB2ROLLFORWARD_QUERY before rolling forward any log files.

The transactions contained in the log files are reapplied to the database. The log is processed as far forward in time as information is available, or until the time specified by the stop time parameter.

Recovery stops when any one of the following events occurs:

- No more log files are found
- A time stamp in the log file exceeds the completion time stamp specified by the stop time parameter
- An error occurs while reading the log file.

Some transactions might not be recovered. The value returned in lastcommit indicates the time stamp of the last committed transaction that was applied to the database.

If the need for database recovery was caused by application or human error, the user may want to provide a time stamp value in piStopTime, indicating that recovery should be stopped before the time of the error. This applies only to full database roll-forward recovery, and to table space rollforward to a point in time. It also permits recovery to be stopped before a log read error occurs, determined during an earlier failed attempt to recover.

When the rollforward_recovery flag is set to DATABASE, the database is not available for use until roll-forward recovery is terminated. Termination is accomplished by calling the API with a caller action of DB2ROLLFORWARD_STOP or DB2ROLLFORWARD_RFWRD_STOP to bring the database out of roll-forward pending state. If the rollforward_recovery flag is TABLESPACE, the database is available for use. However, the table spaces in SQLB_ROLLFORWARD_PENDING and SQLB_ROLLFORWARD_IN_PROGRESS states will not be available until the API is called to perform table space roll-forward recovery. If rolling forward table spaces to a point in time, the table spaces are placed in backup pending state after a successful rollforward.

When the RollforwardFlags option is set to DB2ROLLFORWARD_LOCAL_TIME, all messages returned to the user will also be in local time. All times are converted on the server, and on the catalog partition, if it is a partitioned database environment. The timestamp string is converted to GMT on the server, so the time is local to the server's time zone, not the client's. If the client is in one time zone and the server in another, the server's local time should be used. This is different from the local time option from the Control Center, which is local to the client. If the timestamp string is close to the time change of the clock due to daylight savings, it is important to know if the stop time is before or after the clock change, and specify it correctly.

db2SetWriteForDB - Suspend or resume I/O writes for database

Sets the database to be I/O write suspended, or resumes I/O writes to disk. I/O writes must be suspended for a database before a split mirror can be taken. To avoid potential problems, keep the same connection to do the write suspension and resumption.

Scope

This API only affects the database partition on which it is executed.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection

Database

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2SetWriteForDB (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);
```

```
typedef struct db2SetWriteDbStruct
{
   db2int32 iOption;
   char *piTablespaceNames;
} db2SetWriteDbStruct;
```

db2SetWriteForDB API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2SetWriteDbStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2SetWriteDbStruct data structure parameters

iOption
Input. Specifies the action. Valid values are:

- DB2_DB_SUSPEND_WRITE
Suspends I/O write to disk.

- DB2_DB_RESUME_WRITE
Resumes I/O write to disk.

piTablespaceNames
Input. Reserved for future use.

sqlabndx - Bind application program to create a package

Invokes the bind utility, which prepares SQL statements stored in the bind file generated by the precompiler, and creates a package that is stored in the database.

Scope

This API can be called from any database partition server in db2nodes.cfg. It updates the database catalogs on the catalog partition. Its effects are visible to all database partition servers.

Authorization

One of the following authorizations:

- *dbadm* authority
- If EXPLAIN ONLY is specified, EXPLAIN authority or an authority that implicitly includes EXPLAIN is sufficient.
- If a package does not exist, BINDADD authority and:
  - If the schema name of the package does not exist, IMPLICIT_SCHEMA authority on the database.
  - If the schema name of the package does exist, CREATEIN privilege on the schema.
- If the package exists, one of the following privileges:
  - ALTERIN privilege on the schema
  - BIND privilege on the package

In addition, if capturing explain information using the EXPLAIN or the EXPLSNAP clause, one of the following authorizations is required:

- INSERT privilege on the explain tables
- DATAACCESS authority

The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements.

Required connection

Database

API include file

```
sql.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlabndx (
        _SQLOLDCHAR * pBindFileName,
        _SQLOLDCHAR * pMsgFileName,
        struct sqlopt * pBindOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgbndx (
        unsigned short MsgFileNameLen,
        unsigned short BindFileNameLen,
        struct sqlca * pSqlca,
        struct sqlopt * pBindOptions,
        _SQLOLDCHAR * pMsgFileName,
        _SQLOLDCHAR * pBindFileName);
```

sqlabndx API parameters

pBindFileName

Input. A string containing the name of the bind file, or the name of a file containing a list of bind file names. The bind file names must contain the extension .bnd. A path for these files can be specified.

Precede the name of a bind list file with the at sign (@). For example, a fully qualified bind list file name might be:

```
/u/user1/bnd/@all.lst
```

The bind list file should contain one or more bind file names, and must have the extension .lst.

Precede all but the first bind file name with a plus symbol (+). The bind file names might be on one or more lines. For example, the bind list file `all.lst` might contain:

```
mybind1.bnd+mybind2.bnd+
mybind3.bnd+
mybind4.bnd
```

Path specifications on bind file names in the list file can be used. If no path is specified, the database manager takes path information from the bind list file.

pMsgFileName

Input. A string containing the destination for error, warning, and

informational messages. Can be the path and the name of an operating system file, or a standard device. If a file already exists, it is overwritten. If it does not exist, a file is created.

pBindOptions
Input. A structure used to pass bind options to the API. For more information about this structure, see SQLOPT.

pSqlca
Output. A pointer to the sqlca structure.

sqlgbndx API-specific parameters

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages. Can be the path and the name of an operating system file, or a standard device. If a file already exists, it is overwritten. If it does not exist, a file is created.

BindFileNameLen
Input. Length in bytes of the pBindFileName parameter.

Usage notes

Binding can be done as part of the precompile process for an application program source file, or as a separate step at a later time. Use BIND when binding is performed as a separate process.

The name used to create the package is stored in the bind file, and is based on the source file name from which it was generated (existing paths or extensions are discarded). For example, a precompiled source file called myapp.sqc generates a default bind file called myapp.bnd and a default package name of MYAPP. (However, the bind file name and the package name can be overridden at precompile time by using the SQL_BIND_OPT and the SQL_PKG_OPT options of sqlaprep.)

BIND executes under the transaction that the user has started. After performing the bind, BIND issues a COMMIT (if bind is successful) or a ROLLBACK (if bind is unsuccessful) operation to terminate the current transaction and start another one.

Binding halts if a fatal error or more than 100 errors occur. If a fatal error occurs during binding, BIND stops binding, attempts to close all files, and discards the package.

Binding application programs have prerequisite requirements and restrictions beyond the scope of this manual. For example, an application cannot be bound from a Version 8 client to a Version 8 server, and then executed against a Version 7 server.

The Bind option types and values are defined in sql.h.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqlbftpq - Fetch the query data for rows in a table space

Fetches a specified number of rows of table space query data, each row consisting of data for a table space.

Important: This command or API has been deprecated and might be removed in a future release. You can use the MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions instead which return more information. For more information, see the "LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Scope

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint
- sysmon
- dbadm
- load

Required connection

Database

API include file

```
sqlutil.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlbftpq (
   struct sqlca * pSqlca,
   sqluint32 MaxTablespaces,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 * pNumTablespaces);

SQL_API_RC SQL_API_FN
  sqlgftpq (
   struct sqlca * pSqlca,
   sqluint32 MaxTablespaces,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 * pNumTablespaces);
```

sqlbftpq API parameters

pSqlca
Output. A pointer to the sqlca structure.

MaxTablespaces
Input. The maximum number of rows of data that the user allocated output area (pointed to by pTablespaceData) can hold.

pTablespaceData
Input and output. Pointer to the output area, a structure for query data. For more information about this structure, see SQLB-TBSPQRY-DATA. The caller of this API must:

- Allocate space for MaxTablespaces of these structures
- Initialize the structures
- Set TBSPQVER in the first structure to SQLB_TBSPQRY_DATA_ID
- Set pTablespaceData to point to this space. The API will use this space to return the table space data.

pNumTablespaces
Output. Number of rows of output returned.

Usage notes

The user is responsible for allocating and freeing the memory pointed to by the pTablespaceData parameter. This API can only be used after a successful sqlbotsq call. It can be invoked repeatedly to fetch the list generated by sqlbotsq.

sqlbmtsq - Get the query data for all table spaces

Provides a one-call interface to the table space query data. The query data for all table spaces in the database is returned in an array.

Important: This command or API has been deprecated and might be removed in a future release. You can use the MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions instead which return more information. For more information, see the "LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Scope

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint
- sysmon
- dbadm
- load

Required connection

Database

API include file

`sqlutil.h`

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlbmtsq (
   struct sqlca * pSqlca,
   sqluint32 * pNumTablespaces,
   struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
   sqluint32 reserved1,
   sqluint32 reserved2);

SQL_API_RC SQL_API_FN
  sqlgmtsq (
   struct sqlca * pSqlca,
   sqluint32 * pNumTablespaces,
   struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
   sqluint32 reserved1,
   sqluint32 reserved2);
```

sqlbmtsq API parameters

pSqlca
Output. A pointer to the sqlca structure.

pNumTablespaces
Output. The total number of table spaces in the connected database.

pppTablespaceData
Output. The caller supplies the API with the address of a pointer. The space for the table space query data is allocated by the API, and a pointer to that space is returned to the caller. On return from the call, the pointer points to an array of SQLB_TBSPQRY_DATA pointers to the complete set of table space query data.

reserved1
Input. Always SQLB_RESERVED1.

reserved2
Input. Always SQLB_RESERVED2.

Usage notes

This API uses the lower level services, namely:

- sqlbotsq
- sqlbftpq
- sqlbctsq

to get all of the table space query data at once.

If sufficient memory is available, this function returns the number of table spaces, and a pointer to the memory location of the table space query data. It is the user's responsibility to free this memory with a call to sqlefmem.

If sufficient memory is not available, this function simply returns the number of table spaces, and no memory is allocated. If this should happen, use sqlbotsq, sqlbftpq, and sqlbctsq, to fetch less than the whole list at once.

sqlbotcq - Open a table space container query

Prepares for a table space container query operation, and returns the number of containers currently in the table space.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint
- sysmon
- dbadm

Required connection

Database

API include file

```
sqlutil.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlbotcq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   sqluint32 * pNumContainers);

SQL_API_RC SQL_API_FN
  sqlgotcq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   sqluint32 * pNumContainers);
```

sqlbotcq API parameters

pSqlca
Output. A pointer to the sqlca structure.

TablespaceId
Input. ID of the table space for which container data is desired. If the special identifier SQLB_ALL_TABLESPACES (in sqlutil.h) is specified, a complete list of containers for the entire database is produced.

pNumContainers
Output. The number of containers in the specified table space.

Usage notes

This API is normally followed by one or more calls to sqlbftcq, and then by one call to sqlbctcq.

An application can use the following APIs to fetch information about containers in use by table spaces:

- sqlbtcq

  Fetches a complete list of container information. The API allocates the space required to hold the information for all the containers, and returns a pointer to this information. Use this API to scan the list of containers for specific information. Using this API is identical to calling the three APIs below (sqlbotcq, sqlbftcq, sqlbctcq), except that this API automatically allocates the memory for the output information. A call to this API must be followed by a call to sqlefmem to free the memory.

- sqlbotcq
- sqlbftcq
- sqlbctcq

  These three APIs function like an SQL cursor, in that they use the OPEN/FETCH/CLOSE paradigm. The caller must provide the output area for the fetch. Unlike an SQL cursor, only one table space container query can be active at a time. Use this set of APIs to scan the list of table space containers for specific information. These APIs allows the user to control the memory requirements of an application (compared with sqlbtcq).

When sqlbotcq is called, a snapshot of the current container information is formed in the agent servicing the application. If the application issues a second table space container query call (sqlbtcq or sqlbotcq), this snapshot is replaced with refreshed information.

No locking is performed, so the information in the buffer may not reflect changes made by another application after the snapshot was generated. The information is not part of a transaction.

There is one snapshot buffer for table space queries and another for table space container queries. These buffers are independent of one another.

sqlbstpq - Get information about a single table space

Retrieves information about a single currently defined table space.

Important: This command or API has been deprecated and might be removed in a future release. You can use the MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions instead which return more information. For more information, see the "LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Scope

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint
- sysmon
- dbadm
- load

Required connection

Database

API include file

```
sqlutil.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlbstpq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 reserved);

SQL_API_RC SQL_API_FN
  sqlgstpq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 reserved);
```

sqlbstpq API parameters

pSqlca
Output. A pointer to the sqlca structure.

TablespaceId
Input. Identifier for the table space which is to be queried.

pTablespaceData
Input and output. Pointer to a user-supplied SQLB_TBSPQRY_DATA structure where the table space information will be placed upon return. The caller of this API must initialize the structure and set TBSPQVER to SQLB_TBSPQRY_DATA_ID (in sqlutil).

reserved
Input. Always SQLB_RESERVED1.

Usage notes

This API retrieves information about a single table space if the table space identifier to be queried is known. This API provides an alternative to the more expensive OPEN TABLESPACE QUERY, FETCH, and CLOSE combination of APIs, which must be used to scan for the desired table space when the table space identifier is not known in advance. The table space IDs can be found in the system catalogs. No agent snapshot is taken; since there is only one entry to return, it is returned directly.

sqle_activate_db - Activate database

Activates the specified database and starts up all necessary database services, so that the database is available for connection and use by any application.

Scope

This API activates the specified database on all database partition servers. If one or more of these database partition servers encounters an error during activation of the database, a warning is returned. The database remains activated on all database partition servers on which the API has succeeded.

Note: If it is the coordinator partition or the catalog partition that encounters the error, the API returns a negative sqlcode, and the database will not be activated on any database partition server.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection

None. Applications invoking ACTIVATE DATABASE cannot have any existing database connections.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqle_activate_db (
        char * pDbAlias,
        char * pUserName,
        char * pPassword,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlg_activate_db (
        unsigned short DbAliasLen,
        unsigned short UserNameLen,
        unsigned short PasswordLen,
        char * pDbAlias,
        char * pUserName,
        char * pPassword,
        void * pReserved,
        struct sqlca * pSqlca);
```

sqle_activate_db API parameters

pDbAlias
Input. Pointer to the database alias name.

pUserName
Input. Pointer to the user ID starting the database. Can be NULL.

pPassword
Input. Pointer to the password for the user name. Can be NULL, but must be specified if a user name is specified.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the sqlca structure.

sqlg_activate_db API-specific parameters

DbAliasLen
Input. A 2-byte unsigned integer representing the length of the database alias name in bytes.

UserNameLen
Input. A 2-byte unsigned integer representing the length of the user name in bytes. Set to zero if no user name is supplied.

PasswordLen
Input. A 2-byte unsigned integer representing the length of the password in bytes. Set to zero if no password is supplied.

Usage notes

If a database has not been started, and a DB2 CONNECT TO (or an implicit connect) is encountered in an application, the application must wait while the database manager starts up the required database. In such cases, this first application spends time on database initialization before it can do any work. However, once the first application has started a database, other applications can simply connect and use it.

Database administrators can use ACTIVATE DATABASE to start up selected databases. This eliminates any application time spent on database initialization.

Databases initialized by ACTIVATE DATABASE can only be shut down by sqle_deactivate_db, or by db2InstanceStop. To obtain a list of activated databases, call db2GetSnapshot.

If a database was started by a DB2 CONNECT TO (or an implicit connect) and subsequently an ACTIVATE DATABASE is issued for that same database, then DEACTIVATE DATABASE must be used to shut down that database.

ACTIVATE DATABASE behaves in a similar manner to a DB2 CONNECT TO (or an implicit connect) when working with a database requiring a restart (for example, database in an inconsistent state). The database will be restarted before it can be initialized by ACTIVATE DATABASE.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqle_deactivate_db - Deactivate database

Stops the specified database.

Scope

In a partitioned database environment, this API deactivates the specified database on all database partition servers. If one or more of these database partition servers encounters an error, a warning is returned. The database will be successfully deactivated on some database partition servers, but may remain activated on the database partition servers encountering the error.

Note: If it is the coordinator partition or the catalog partition that encounters the error, the API returns a negative sqlcode, and the database will not be reactivated on any database partition server on which it was deactivated.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection

None. Applications invoking DEACTIVATE DATABASE cannot have any existing database connections.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqle_deactivate_db (
        char * pDbAlias,
        char * pUserName,
        char * pPassword,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlg_deactivate_db (
        unsigned short DbAliasLen,
        unsigned short UserNameLen,
        unsigned short PasswordLen,
        char * pDbAlias,
        char * pUserName,
        char * pPassword,
        void * pReserved,
        struct sqlca * pSqlca);
```

sqle_deactivate_db API parameters

pDbAlias
Input. Pointer to the database alias name.

pUserName
Input. Pointer to the user ID stopping the database. Can be NULL.

pPassword
Input. Pointer to the password for the user name. Can be NULL, but must be specified if a user name is specified.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the sqlca structure.

sqlg_deactivate_db API-specific parameters

DbAliasLen
Input. A 2-byte unsigned integer representing the length of the database alias name in bytes.

UserNameLen
Input. A 2-byte unsigned integer representing the length of the user name in bytes. Set to zero if no user name is supplied.

PasswordLen
Input. A 2-byte unsigned integer representing the length of the password in bytes. Set to zero if no password is supplied.

Usage notes

Databases initialized by ACTIVATE DATABASE can only be shut down by DEACTIVATE DATABASE. db2InstanceStop automatically stops all activated databases before stopping the database manager. If a database was initialized by ACTIVATE DATABASE, the last DB2 CONNECT RESET statement (counter equal 0) will not shut down the database; DEACTIVATE DATABASE must be used.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqleaddn - Add a database partition to the partitioned database environment

Adds a database partition to a database partition server.

Scope

This API only affects the database partition server on which it is executed.

Authorization

One of the following:

- sysadm
- sysctrl

Required connection

None

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqleaddn (
        void * pAddNodeOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
```

```
sqlgaddn (
      unsigned short addnOptionsLen,
      struct sqlca * pSqlca,
      void * pAddNodeOptions);
```

sqleaddn API parameters

pAddNodeOptions
Input. A pointer to the optional sqle_addn_options structure. This structure is used to specify the source database partition server, if any, of the system temporary table space definitions for all database partitions to be created. If not specified (that is, a NULL pointer is specified), the system temporary table space definitions will be the same as those for the catalog partition.

pSqlca
Output. A pointer to the sqlca structure.

sqlgaddn API-specific parameters

addnOptionsLen
Input. A 2-byte unsigned integer representing the length of the optional sqle_addn_options structure in bytes.

Usage notes

This API should only be used if a database partition server is added to an environment that has one database and that database is not cataloged at the time of the add partition operation. In this situation, because the database is not cataloged, the add partition operation does not recognize the database, and does not create a database partition for the database on the new database partition server. Any attempt to connect to the database partition on the new database partition server results in an error. The database must first be cataloged before the sqleaddn API can be used to create the database partition for the database on the new database partition server.

This API should not be used if the environment has more than one database and at least one of the databases is cataloged at the time of the add partition operation. In this situation, use the sqlecran API to create a database partition for each database that was not cataloged at the time of the add partition operation. Each uncataloged database must first be cataloged before the sqlecran API can be used to create the database partition for the database on the new database partition server.

Before adding a new database partition, ensure that there is sufficient storage for the containers that must be created.

The add node operation creates an empty database partition on the new database partition server for every database that exists in the instance. The configuration parameters for the new database partitions are set to the default value.

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

If an add node operation fails while creating a database partition locally, it enters a clean-up phase, in which it locally drops all databases that have been created. This means that the database partitions are removed only from the database partition server being added (that is, the local database partition server). Existing database

partitions remain unaffected on all other database partition servers. If this fails, no further clean up is done, and an error is returned.

The database partitions on the new database partition server cannot be used to contain user data until after the ALTER DATABASE PARTITION GROUP statement has been used to add the database partition server to a database partition group.

This API will fail if a create database or a drop database operation is in progress. The API can be called again when the operation has completed.

To determine whether or not a database is enabled for automatic storage, the sqleaddn API has to communicate with the catalog partition for each of the databases in the instance. If automatic storage is enabled then 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 sqleaddn API may have to communicate with another database partition server in the partitioned database environment in order to retrieve the table space definitions. The **start_stop_time** database manager configuration parameter is used to specify the time, in minutes, by which the other database partition server must respond with the automatic storage and table space definitions. If this time is exceeded, the API fails. Increase the value of **start_stop_time**, and call the API again.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqlecadb - Catalog a database in the system database directory

Stores database location information in the system database directory. The database can be located either on the local workstation or on a remote database partition server.

Scope

This API affects the system database directory. In a partitioned database environment, when cataloging a local database into the system database directory, this API must be called from a database partition server where the database resides.

Authorization

One of the following:

- SYSADM
- SYSCTRL

Required connection

None

API include file

`sqlenv.h`

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlecadb (
        _SQLOLDCHAR * pDbName,
        _SQLOLDCHAR * pDbAlias,
        unsigned char Type,
        _SQLOLDCHAR * pNodeName,
        _SQLOLDCHAR * pPath,
        _SQLOLDCHAR * pComment,
        unsigned short Authentication,
        _SQLOLDCHAR * pPrincipal,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgcadb (
        unsigned short PrinLen,
        unsigned short CommentLen,
        unsigned short PathLen,
        unsigned short NodeNameLen,
        unsigned short DbAliasLen,
        unsigned short DbNameLen,
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pPrinName,
        unsigned short Authentication,
        _SQLOLDCHAR * pComment,
        _SQLOLDCHAR * pPath,
        _SQLOLDCHAR * pNodeName,
        unsigned char Type,
        _SQLOLDCHAR * pDbAlias,
        _SQLOLDCHAR * pDbName);
```

sqlecadb API parameters

pDbName
Input. A string containing the database name.

pDbAlias
Input. A string containing an alias for the database.

Type Input. A single character that designates whether the database is indirect, remote, or is cataloged via DCE. Valid values (defined in `sqlenv.h`) are:

SQL_INDIRECT
Specifies that the database resides at this instance.

SQL_REMOTE
Specifies that the database resides at another instance.

SQL_DCE
Specifies that the database is cataloged via DCE.

pNodeName
Input. A string containing the name of the database partition where the database is located. May be NULL.

Note: If neither **pPath** nor **pNodeName** is specified, the database is assumed to be local, and the location of the database is assumed to be that specified in the database manager configuration parameter **dftdbpath**.

pPath

Input. A string which, on Linux and UNIX systems, specifies the name of the path on which the database being cataloged resides. Maximum length is 215 characters.

On the Windows operating system, this string specifies the letter of the drive on which the database being cataloged resides.

If a NULL pointer is provided, the default database path is assumed to be that specified by the database manager configuration parameter **dftdbpath**.

pComment

Input. A string containing an optional description of the database. A null string indicates no comment. The maximum length of a comment string is 30 characters.

Authentication

Input. Contains the authentication type specified for the database. Authentication is a process that verifies that the user is who he/she claims to be. Access to database objects depends on the user's authentication. Valid values (from `sqlenv.h`) are:

SQL_AUTHENTICATION_SERVER
Specifies that authentication takes place on the database partition server containing the target database.

SQL_AUTHENTICATION_CLIENT
Specifies that authentication takes place on the database partition server where the application is invoked.

SQL_AUTHENTICATION_KERBEROS
Specifies that authentication takes place using Kerberos Security Mechanism.

SQL_AUTHENTICATION_NOT_SPECIFIED
Authentication not specified.

SQL_AUTHENTICATION_SVR_ENCRYPT
Specifies that authentication takes place on the database partition server containing the target database, and that the authentication password is to be encrypted.

SQL_AUTHENTICATION_DATAENC
Specifies that authentication takes place on the database partition server containing the target database, and that connections must use data encryption.

SQL_AUTHENTICATION_GSSPLUGIN
Specifies that authentication takes place using an external GSS API-based plug-in security mechanism.

SQL_AUTHENTICATION_SVRENC_AESO
Specifies that authentication takes place on the database partition server containing the target database, and that the authentication userid and password are to be encrypted using an Advanced Encryption Standard (AES) encryption algorithm.

This parameter can be set to SQL_AUTHENTICATION_NOT_SPECIFIED, except when cataloging a database that resides on a DB2 Version 1 server.

Specifying the authentication type in the database catalog results in a performance improvement during a connect.

pPrincipal

Input. A string containing the principal name of the DB2 server on which the database resides. This value should only be specified when **authentication** is SQL_AUTHENTICATION_KERBEROS.

pSqlca
Output. A pointer to the sqlca structure.

sqlgcadb API-specific parameters

PrinLen
Input. A 2-byte unsigned integer representing the length in bytes of the principal name. Set to zero if no principal is provided. This value should be nonzero only when **authentication** is specified as SQL_AUTHENTICATION_KERBEROS.

CommentLen
Input. A 2-byte unsigned integer representing the length in bytes of the comment. Set to 0 if no comment is provided.

PathLen
Input. A 2-byte unsigned integer representing the length in bytes of the path of the local database directory. Set to 0 if no path is provided.

NodeNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the node name. Set to 0 if no node name is provided.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

DbNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the database name.

pPrinName
Input. A string containing the principal name of the DB2 server on which the database resides. This value should only be specified when **authentication** is SQL_AUTHENTICATION_KERBEROS.

Usage notes

Use CATALOG DATABASE to catalog databases located on local or remote nodes, recatalog databases that were uncataloged previously, or maintain multiple aliases for one database (regardless of database location).

DB2 automatically catalogs databases when they are created. It catalogs an entry for the database in the local database directory, and another entry in the system database directory. If the database is created from a remote client (or a client which is executing from a different instance on the same machine), an entry is also made in the system database directory at the client instance.

Databases created at the current instance (as defined by the value of the **DB2INSTANCE** environment variable) are cataloged as indirect. Databases created at other instances are cataloged as remote (even if they physically reside on the same machine).

CATALOG DATABASE automatically creates a system database directory if one does not exist. The system database directory is stored on the path that contains the database manager instance that is being used. The system database directory is maintained outside of the database. Each entry in the directory contains:

- Alias
- Authentication type

- Comment
- Database
- Entry type
- Local database directory (when cataloging a local database)
- Node name (when cataloging a remote database)
- Release information

If a database is cataloged with the **type** parameter set to SQL_INDIRECT, the value of the **authentication** parameter provided will be ignored, and the authentication in the directory will be set to SQL_AUTHENTICATION_NOT_SPECIFIED.

If directory caching is enabled, database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (db2stop) and then restart (db2start) the database manager. To refresh the directory cache for another application, stop and then restart that application.

REXX API syntax

```
CATALOG DATABASE dbname [AS alias] [ON path|AT NODE nodename]
[AUTHENTICATION authentication] [WITH "comment"]
CATALOG GLOBAL DATABASE db_global_name AS alias
USING DIRECTORY {DCE} [WITH "comment"]
```

REXX API parameters

dbname
Name of the database to be cataloged.

alias Alternate name for the database. If an alias is not specified, the database name is used as the alias.

path Path on which the database being cataloged resides.

nodename
Name of the remote workstation where the database being cataloged resides.

Note: If neither **path** nor **nodename** is specified, the database is assumed to be local, and the location of the database is assumed to be that specified in the database manager configuration parameter **dftdbpath**.

authentication
Place where authentication is to be done. Valid values are:

SERVER
Authentication occurs at the database partition server containing the target database. This is the default.

CLIENT
Authentication occurs at the database partition server where the application is invoked.

KERBEROS
Specifies that authentication takes place using Kerberos Security Mechanism.

NOT_SPECIFIED
Authentication not specified.

SVR_ENCRYPT
Specifies that authentication takes place on the database partition server containing the target database, and that the authentication userid and password are to be encrypted.

DATAENC
Specifies that authentication takes place on the database partition server containing the target database, and that connections must use data encryption.

GSSPLUGIN
Specifies that authentication takes place using an external GSS API-based plug-in security mechanism.

SQL_AUTHENTICATION_SVRENC_AESO
Specifies that authentication takes place on the database partition server containing the target database, and that the authentication userid and password are to be encrypted using an AES encryption algorithm.

comment
Describes the database or the database entry in the system database directory. The maximum length of a comment string is 30 characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by double quotation marks.

db_global_name
The fully qualified name that uniquely identifies the database in the DCE name space.

DCE The global directory service being used.

REXX examples

```
call SQLDBS 'CATALOG GLOBAL DATABASE /.../cell1/subsys/database/DB3
AS dbtest USING DIRECTORY DCE WITH "Sample Database"'
```

sqlecrea - Create database

Initializes a new database with an optional user-defined collating sequence, creates the three initial table spaces, creates the system tables, and allocates the recovery log.

Scope

In a partitioned database environment, this API affects all database partition servers that are listed in the `db2nodes.cfg` file.

The database partition server from which this API is called becomes the catalog partition for the new database.

Authorization

One of the following:

- sysadm
- sysctrl

Required connection

Instance. To create a database at another (remote) node, it is necessary to first attach to that node. A database connection is temporarily established by this API during processing.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlecrea (
        char * pDbName,
        char * pLocalDbAlias,
        char * pPath,
        struct sqledbdesc * pDbDescriptor,
        SQLEDBTERRITORYINFO * pTerritoryInfo,
        char Reserved2,
        void * pDbDescriptorExt,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgcrea (
        unsigned short PathLen,
        unsigned short LocalDbAliasLen,
        unsigned short DbNameLen,
        struct sqlca * pSqlca,
        void * pReserved1,
        unsigned short Reserved2,
        SQLEDBTERRITORYINFO * pTerritoryInfo,
        struct sqledbdesc * pDbDescriptor,
        char * pPath,
        char * pLocalDbAlias,
        char * pDbName);
```

sqlecrea API parameters

pDbName
Input. A string containing the database name. This is the database name that will be cataloged in the system database directory. Once the database has been successfully created in the server's system database directory, it is automatically cataloged in the system database directory with a database alias identical to the database name. Must not be NULL.

pLocalDbAlias
Input. A string containing the alias to be placed in the client's system database directory. Can be NULL. If no local alias is specified, the database name is the default.

pPath Input. On Linux and UNIX systems, specifies the path on which to create the database. If a path is not specified, the database is created on the default database path specified in the database manager configuration file (dftdbpath parameter). On the Windows operating system, specifies the letter of the drive on which to create the database. Can be NULL.

Note: For partitioned database environments, a database should not be created in an NFS-mounted directory. If a path is not specified, ensure that the dftdbpath database manager configuration parameter is not set to an NFS-mounted path (for example, on UNIX based systems, it should not specify the $HOME directory of the instance owner). The path specified for this API in a partitioned database environment cannot be a relative path.

pDbDescriptor
Input. A pointer to the database description block that is used when creating the database. The database description block can be used by you to supply values that are permanently stored in the configuration file of the database.

The supplied values are a collating sequence, a database comment, or a table space definition. The supplied value can be NULL if you do not want to supply any values. For information about the values that can be supplied through this parameter, see the SQLEDBDESC data structure topic.

pTerritoryInfo
Input. A pointer to the sqledbterritoryinfo structure, containing the locale and the code set for the database. Can be NULL. The default code set for a database is UTF-8 (Unicode). If a particular code set and territory is needed for a database, the desired code set and territory should be specified via the sqledbterritoryinfo structure. If this field is NULL, then one of the following is allowed as a collation value for the database (sqlcode 1083): NULL, SQL_CS_SYSTEM, SQL_CS_IDENTITY_16BIT, SQL_CS_UCA400_NO, SQL_CS_UCA400_LTH, SQL_CS_UCA400_LSK, or SQL_CS_UNICODE.

Reserved2
Input. Reserved for future use.

pDbDescriptorExt
Input. This parameter refers to an extended database description block (sqledbdescext) that is used when creating the database. The extended database description block controls automatic storage for a database, chooses a default page size for the database, and specifies values for new table space attributes that have been introduced. If set to null or zero, a default page size of 4 096 bytes is chosen for the database and automatic storage is enabled.

pSqlca
Output. A pointer to the sqlca structure.

sqlgcrea API-specific parameters

PathLen
Input. A 2-byte unsigned integer representing the length of the path in bytes. Set to zero if no path is provided.

LocalDbALiasLen
Input. A 2-byte unsigned integer representing the length of the local database alias in bytes. Set to zero if no local alias is provided.

DbNameLen
Input. A 2-byte unsigned integer representing the length of the database name in bytes.

Usage notes

CREATE DATABASE:

- Creates a database in the specified subdirectory. In a partitioned database environment, creates the database on all database partition servers listed in `db2nodes.cfg`, and creates a $DB2INSTANCE/NODExxxx directory under the specified subdirectory at each database partition server, where xxxx represents

the local database partition server number. In a single-partition environment, creates a $DB2INSTANCE/NODE0000 directory under the specified subdirectory.

- Creates the system catalog tables and recovery log.
- Catalogs the database in the following database directories:
  - server's local database directory on the path indicated by pPath or, if the path is not specified, the default database path defined in the database manager system configuration file. A local database directory resides on each file system that contains a database.
  - server's system database directory for the attached instance. The resulting directory entry will contain the database name and a database alias.

    If the API was called from a remote client, the client's system database directory is also updated with the database name and an alias.
- Creates a system or a local database directory if neither exists. If specified, the comment and code set values are placed in both directories.
- Stores the specified code set, territory, and collating sequence. A flag is set in the database configuration file if the collating sequence consists of unique weights, or if it is the identity sequence.
- Creates the schemata called SYSCAT, SYSFUN, SYSIBM, and SYSSTAT with SYSIBM as the owner. The database partition server on which this API is called becomes the catalog partition for the new database. Two database partition groups are created automatically: IBMDEFAULTGROUP and IBMCATGROUP.
- Binds the previously defined database manager bind files to the database (these are listed in db2ubind.lst). If one or more of these files do not bind successfully, sqlecrea returns a warning in the SQLCA, and provides information about the binds that failed. If a bind fails, the user can take corrective action and manually bind the failing file. The database is created in any case. A schema called NULLID is implicitly created when performing the binds with CREATEIN privilege granted to PUBLIC, if the RESTRICTIVE option is not selected.
- Creates SYSCATSPACE, TEMPSPACE1, and USERSPACE1 table spaces. The SYSCATSPACE table space is only created on the catalog partition. All database partitions have the same table space definitions.
- Grants the following:
  - DBADM, CONNECT, CREATETAB, BINDADD, CREATE_NOT_FENCED, IMPLICIT_SCHEMA, and LOAD authorities to the database creator
  - CONNECT, CREATETAB, BINDADD, and IMPLICIT_SCHEMA authorities to PUBLIC
  - USE privilege on the USERSPACE1 table space to PUBLIC
  - SELECT privilege on each system catalog to PUBLIC
  - BIND and EXECUTE privilege to PUBLIC for each successfully bound utility
  - EXECUTE WITH GRANT privilege to PUBLIC on all functions in the SYSFUN schema.
  - EXECUTE privilege to PUBLIC on all procedures in SYSIBM schema.

Note: If the RESTRICTIVE option is present, it causes the RESTRICT_ACCESS database configuration parameter to be set to YES and no privileges or authorities are automatically granted to PUBLIC. For more detailed information, see the RESTRICTIVE option of the CREATE DATABASE command.

With dbadm authority, one can grant these privileges to (and revoke them from) other users or PUBLIC. If another administrator with sysadm or dbadm authority over the database revokes these privileges, the database creator nevertheless retains them.

In a partitioned database environment, the database manager creates a subdirectory, $DB2INSTANCE/NODExxxx, under the specified or default path on all database partition servers. The xxxx is the node number as defined in the `db2nodes.cfg` file (that is, node 0 becomes NODE0000). Subdirectories SQL00001 through SQLnnnnn will reside on this path. This ensures that the database objects associated with different database partition servers are stored in different directories (even if the subdirectory $DB2INSTANCE under the specified or default path is shared by all database partition servers).

On Windows and AIX operating systems, the length of the code set name is limited to a maximum of 9 characters. For example, specify a code set name such as ISO885915 instead of ISO8859-15.

The sqlecrea API accepts a data structure called the Database Descriptor Block (SQLEDBDESC). You can define your own collating sequence within this structure.

Note: You can only define your own collating sequence for a single-byte database.

To specify a collating sequence for a database:

- Pass the desired SQLEDBDESC structure, or
- Pass a NULL pointer. The collating sequence of the operating system (based on the current locale code and the code page) is used. This is the same as specifying SQLDBCSS equal to SQL_CS_SYSTEM (0).

Execution of the CREATE DATABASE command will fail if the application is already connected to a database.

If the database description block structure is not set correctly, an error message is returned.

The most prominent value of the database description block must be set to the symbolic value SQLE_DBDESC_2 (defined in sqlenv). The following sample user-defined collating sequences are available in the host language include files:

sqle819a
: If the code page of the database is 819 (ISO Latin/1), this sequence will cause sorting to be performed according to the host CCSID 500 (EBCDIC International).

sqle819b
: If the code page of the database is 819 (ISO Latin/1),this sequence will cause sorting to be performed according to the host CCSID 037 (EBCDIC US English).

sqle850a
: If the code page of the database is 850 (ASCII Latin/1), this sequence will cause sorting to be performed according to the host CCSID 500 (EBCDIC International).

sqle850b
If the code page of the database is 850 (ASCII Latin/1), this sequence will cause sorting to be performed according to the host CCSID 037 (EBCDIC US English).

sqle932a
If the code page of the database is 932 (ASCII Japanese), this sequence will cause sorting to be performed according to the host CCSID 5035 (EBCDIC Japanese).

sqle932b
If the code page of the database is 932 (ASCII Japanese), this sequence will cause sorting to be performed according to the host CCSID 5026 (EBCDIC Japanese).

The collating sequence specified during database creation cannot be changed later. It determines how character strings are compared. This affects the structure of indexes as well as the results of queries. In a Unicode database, the catalog tables and views are always created with the IDENTITY collation, regardless of the collation specified in the create database call. In a non-Unicode database, the catalog tables and views are created with the database collation.

Use sqlecadb to define different alias names for the new database.

The Configuration Advisor is called by default during the database creation process unless specifically told not to do so.

REXX API syntax

```
CREATE DATABASE dbname [ON path] [ALIAS dbalias]
[USING CODESET codeset TERRITORY territory]
[COLLATE USING {SYSTEM | IDENTITY | USER :udcs}]
[NUMSEGS numsegs] [DFT_EXTENT_SZ dft_extentsize]
[CATALOG TABLESPACE <tablespace_definition>]
[USER TABLESPACE <tablespace_definition>]
[TEMPORARY TABLESPACE <tablespace_definition>]
[WITH comment]


Where <tablespace_definition> stands for:
MANAGED BY {
SYSTEM USING :SMS_string |
DATABASE USING :DMS_string }
[ EXTENTSIZE number_of_pages ]
[ PREFETCHSIZE number_of_pages ]
[ OVERHEAD number_of_milliseconds ]
[ TRANSFERRATE number_of_milliseconds ]
```

REXX API parameters

dbname
Name of the database.

dbalias
Alias of the database.

path Path on which to create the database. If a path is not specified, the database is created on the default database path specified in the database manager configuration file (dftdbpath configuration parameter).

Note: For partitioned database environments, a database should not be created in an NFS-mounted directory. If a path is not specified, ensure that

the dftdbpath database manager configuration parameter is not set to an NFS-mounted path (for example, on UNIX based systems, it should not specify the $HOME directory of the instance owner). The path specified for this API in a partitioned database environment cannot be a relative path.

codeset
Code set to be used for data entered into the database.

territory
Territory code (locale) to be used for data entered into the database.

SYSTEM
For non-Unicode databases, this is the default option, with the collating sequence based on the database territory. For Unicode databases, this option is equivalent to the IDENTITY option.

IDENTITY
Identity collating sequence, in which strings are compared byte for byte. This is the default for Unicode databases.

USER udcs
The collating sequence is specified by the calling application in a host variable containing a 256-byte string defining the collating sequence.

numsegs
Number of directories (table space containers) that will be created and used to store the database table files for any default SMS table spaces.

dft_extentsize
Specifies the default extent size for table spaces in the database.

SMS_string
A compound REXX host variable identifying one or more containers that will belong to the table space, and where the table space data will be stored. In the following, XXX represents the host variable name. Note that each of the directory names cannot exceed 254 bytes in length.

XXX.0 Number of directories specified

XXX.1 First directory name for SMS table space

XXX.2 Second directory name for SMS table space

XXX.3 and so on.

DMS_string
A compound REXX host variable identifying one or more containers that will belong to the table space, where the table space data will be stored, container sizes (specified in a number of 4KB pages) and types (file or device). The specified devices (not files) must already exist. In the following, XXX represents the host variable name. Note that each of the container names cannot exceed 254 bytes in length.

XXX.0 Number of strings in the REXX host variable (number of first level elements)

XXX.1.1
Type of the first container (file or device)

XXX.1.2
First file name or device name

XXX.1.3
Size (in pages) of the first container

XXX.2.1
Type of the second container (file or device)

XXX.2.2
Second file name or device name

XXX.2.3
Size (in pages) of the second container

XXX.3.1
and so on.

EXTENTSIZE number_of_pages
Number of 4KB pages that will be written to a container before skipping to the next container.

PREFETCHSIZE number_of_pages
Number of 4KB pages that will be read from the table space when data prefetching is being performed.

OVERHEAD number_of_milliseconds
Number that specifies the I/O controller overhead, disk seek, and latency time in milliseconds.

TRANSFERRATE number_of_milliseconds
Number that specifies the time in milliseconds to read one 4 KB page into memory.

comment
Description of the database or the database entry in the system directory. Do not use a carriage return or line feed character in the comment. Be sure to enclose the comment text in double quotation marks. Maximum size is 30 characters.

sqledpan - Drop a database on a database partition server

Drops a database at a specified database partition server. Can only be run in a partitioned database environment.

Scope

This API only affects the database partition server on which it is called.

Authorization

One of the following:

- sysadm
- sysctrl

Required connection

None. An instance attachment is established for the duration of the call.

API include file

`sqlenv.h`

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqledpan (
        char * pDbAlias,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdpan (
        unsigned short Reserved1,
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        void * pReserved2,
        char * pDbAlias);
```

sqledpan API parameters

pDbAlias
Input. A string containing the alias of the database to be dropped. This name is used to reference the actual database name in the system database directory.

pReserved
Reserved. Should be NULL.

pSqlca
Output. A pointer to the sqlca structure.

sqlgdpan API-specific parameters

Reserved1
Reserved for future use.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

pReserved2
A spare pointer that is set to null or points to zero. Reserved for future use.

Usage notes

Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqledrpd - Drop database

Deletes the database contents and all log files for the database, uncatalogs the database, and deletes the database subdirectory.

Scope

By default, this API affects all database partition servers that are listed in the db2nodes.cfg file.

Authorization

One of the following:

- sysadm
- sysctrl

Required connection

Instance. It is not necessary to call ATTACH before dropping a remote database. If the database is cataloged as remote, an instance attachment to the remote node is established for the duration of the call.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqledrpd (
        _SQLOLDCHAR * pDbAlias,
        _SQLOLDCHAR * pReserved2,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdrpd (
        unsigned short Reserved1,
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pReserved2,
        _SQLOLDCHAR * pDbAlias);
```

sqledrpd API parameters

pDbAlias
Input. A string containing the alias of the database to be dropped. This name is used to reference the actual database name in the system database directory.

pReserved2
A spare pointer that is set to null or points to zero. Reserved for future use.

pSqlca
Output. A pointer to the sqlca structure.

sqlgdrpd API-specific parameters

Reserved1
Reserved for future use.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

Usage notes

The sqledrpd API deletes all user data and log files. If the log files are needed for a roll-forward recovery after a restore operation, the files should be saved prior to calling this API.

The database must not be in use; all users must be disconnected from the database before the database can be dropped.

To be dropped, a database must be cataloged in the system database directory. Only the specified database alias is removed from the system database directory. If other aliases with the same database name exist, their entries remain. If the database being dropped is the last entry in the local database directory, the local database directory is deleted automatically.

If this API is called from a remote client (or from a different instance on the same machine), the specified alias is removed from the client's system database directory. The corresponding database name is removed from the server's system database directory.

REXX API syntax

```
DROP DATABASE dbalias
```

REXX API parameters

dbalias
The alias of the database to be dropped.

sqledrpn - Check whether a database partition server can be dropped

Verifies whether a database partition server is being used by a database. A message is returned, indicating whether the database partition server can be dropped.

Scope

This API only affects the database partition server on which it is issued.

Authorization

One of the following:

- sysadm
- sysctrl

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqledrpn (
        unsigned short Action,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdrpn (
        unsigned short Reserved1,
        struct sqlca * pSqlca,
        void * pReserved2,
        unsigned short Action);
```

sqledrpn API parameters

Action
The action requested. The valid value is: SQL_DROPNODE_VERIFY

pReserved
Reserved. Should be NULL.

pSqlca
Output. A pointer to the sqlca structure.

sqlgdrpn API-specific parameters

Reserved1
Reserved for the length of pReserved2.

pReserved2
A spare pointer that is set to NULL or points to 0. Reserved for future use.

Usage notes

If a message is returned, indicating that the database partition server is not in use, use the db2stop command with DROP NODENUM to remove the entry for the database partition server from the db2nodes.cfg file, which removes the database partition server from the partitioned database environment.

If a message is returned, indicating that the database partition server is in use, the following actions should be taken:

1. The database partition server to be dropped will have a database partition on it for each database in the instance. If any of these database partitions contain data, redistribute the database partition groups that use these database partitions. Redistribute the database partition groups to move the data to database partitions that exist at database partition servers that are not being dropped.
2. After the database partition groups are redistributed, drop the database partition from every database partition group that uses it. To remove a database partition from a database partition group, you can use either the drop node option of the sqludrdt API or the ALTER DATABASE PARTITION GROUP statement.
3. Drop any event monitors that are defined on the database partition server.
4. Rerun sqledrpn to ensure that the database partition at the database partition server is no longer in use.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqlefrce - Force users and applications off the system

Forces local or remote users or applications off the system to allow for maintenance on a server. Attention: If an operation that cannot be interrupted (a database restore, for example) is forced, the operation must be successfully re-executed before the database becomes available.

Scope

This API affects all database partition servers that are listed in the db2nodes.cfg file.

In a partitioned database environment, this API does not have to be issued from the coordinator partition of the application being forced. This API can be issued from any database partition server in the partitioned database environment.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection

Instance. To force users off a remote server, it is necessary to first attach to that server. If no attachment exists, this API is executed locally.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlefrce (
        sqlint32 NumAgentIds,
        sqluint32 * pAgentIds,
        unsigned short ForceMode,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgfrce (
        struct sqlca * pSqlca,
        unsigned short ForceMode,
        sqluint32 * pAgentIds,
        sqlint32 NumAgentIds);
```

sqlefrce API parameters

NumAgentIds

Input. An integer representing the total number of users to be terminated. This number should be the same as the number of elements in the array of agent IDs.

If this parameter is set to SQL_ALL_USERS (defined in sqlenv), all applications with either database connections or instance attachments are forced. If it is set to zero, an error is returned.

pAgentIds

Input. Pointer to an array of unsigned long integers. Each entry describes the agent ID of the corresponding database user.

ForceMode

Input. An integer specifying the operating mode of the sqlefrce API. Only the asynchronous mode is supported. This means that the API does not wait until all specified users are terminated before returning. It returns as soon as the API has been issued successfully, or an error occurs. As a

result, there may be a short interval between the time the force application call completes and the specified users have been terminated.

This parameter must be set to SQL_ASYNCH (defined in sqlenv).

pSqlca
Output. A pointer to the sqlca structure.

Usage notes

The database manager remains active so that subsequent database manager operations can be handled without the need for db2start.

To preserve database integrity, only users who are idling or executing interruptible database operations can be forced off.

After a force command has been issued, the database will still accept requests to connect. Additional forces may be required to completely force all users off. The database system monitor functions are used to gather the agent IDs of the users to be forced.

When the force mode is set to SQL_ASYNCH (the only value permitted), the API immediately returns to the calling application.

Minimal validation is performed on the array of agent IDs to be forced. The user must ensure that the pointer points to an array containing the total number of elements specified. If NumAgentIds is set to SQL_ALL_USERS, the array is ignored.

When a user is forced off, a unit of work rollback is performed to ensure database consistency.

All users that can be forced will be forced. If one or more specified agent IDs cannot be found, sqlcode in the sqlca structure is set to 1230. An agent ID may not be found, for instance, if the user signs off between the time an agent ID is collected and sqlefrce is called. The user that calls this API is never forced off.

Agent IDs are recycled, and are used to force applications some time after being gathered by the database system monitor. When a user signs off, therefore, another user may sign on and acquire the same agent ID through this recycling process, with the result that the wrong user may be forced.

REXX API syntax

```
FORCE APPLICATION {ALL | :agentidarray} [MODE ASYNC]
```

REXX API parameters

ALL All applications will be disconnected. This includes applications that have database connections and applications that have instance attachments.

agentidarray
A compound REXX host variable containing the list of agent IDs to be terminated. In the following, XXX is the name of the host variable:

- XXX.0
Number of agents to be terminated

- XXX.1
First agent ID

- XXX.2
Second agent ID

- XXX.3
and so on.

ASYNC
The only mode currently supported means that sqlefrce does not wait until all specified applications are terminated before returning.

sqlemgdb - Migrate previous version of DB2 database to current version

Converts a previous (Version 8 or higher) version of a DB2 database to the current release. The sqlemgdb and sqlgmgdb APIs are deprecated and will be discontinued in a future release. You should use the new db2DatabaseUpgrade API instead.

Authorization

SYSADM

Required connection

This API establishes a database connection.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlemgdb (
        _SQLOLDCHAR * pDbAlias,
        _SQLOLDCHAR * pUserName,
        _SQLOLDCHAR * pPassword,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgmgdb (
        unsigned short PasswordLen,
        unsigned short UserNameLen,
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pPassword,
        _SQLOLDCHAR * pUserName,
        _SQLOLDCHAR * pDbAlias);
```

sqlemgdb API parameters

pDbAlias
Input. A string containing the alias of the database that is cataloged in the system database directory.

pUserName
Input. A string containing the user name of the application. May be NULL.

pPassword
Input. A string containing the password of the supplied user name (if any). May be NULL.

pSqlca
Output. A pointer to the sqlca structure.

sqlgmgdb API-specific parameters

PasswordLen
Input. A 2-byte unsigned integer representing the length in bytes of the password. Set to zero when no password is supplied.

UserNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero when no user name is supplied.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

Usage notes

This API will only migrate a database to a newer version, and cannot be used to convert a migrated database to its previous version.

The database must be cataloged before migration.

REXX API syntax

```
MIGRATE DATABASE dbalias [USER username USING password]
```

REXX API parameters

dbalias
Alias of the database to be migrated.

username
User name under which the database is to be restarted.

password
Password used to authenticate the user name.

sqlesdeg - Set the maximum runtime intra-partition parallelism level or degree for SQL statements

Sets the maximum run time degree of intra-partition parallelism for SQL statement execution for specified active applications. It has no effect on CREATE INDEX statement execution parallelism.

Scope

This API affects all database partition servers that are listed in the db2nodes.cfg file.

Authorization

One of the following:

- sysadm
- sysctrl

Required connection

Instance. To change the maximum run time degree of parallelism on a remote server, it is first necessary to attach to that server. If no attachment exists, the SET RUNTIME DEGREE statement fails.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlesdeg (
        sqlint32 NumAgentIds,
        sqluint32 * pAgentIds,
        sqlint32 Degree,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgsdeg (
        struct sqlca * pSqlca,
        sqlint32 Degree,
        sqluint32 * pAgentIds,
        sqlint32 NumAgentIds);
```

sqlesdeg API parameters

NumAgentIds
Input. An integer representing the total number of active applications to which the new degree value will apply. This number should be the same as the number of elements in the array of agent IDs.

If this parameter is set to SQL_ALL_USERS (defined in sqlenv), the new degree will apply to all active applications. If it is set to zero, an error is returned.

pAgentIds
Input. Pointer to an array of unsigned long integers. Each entry describes the agent ID of the corresponding application. To list the agent IDs of the active applications, use the db2GetSnapshot API.

Degree
Input. The new value for the maximum run time degree of parallelism. The value must be in the range 1 to 32767.

pSqlca
Output. A pointer to the sqlca structure.

Usage notes

The database system monitor functions are used to gather the agent IDs and degrees of active applications.

Minimal validation is performed on the array of agent IDs. The user must ensure that the pointer points to an array containing the total number of elements specified. If NumAgentIds is set to SQL_ALL_USERS, the array is ignored.

If one or more specified agent IDs cannot be found, the unknown agent IDs are ignored, and the function continues. No error is returned. An agent ID may not be found, for instance, if the user signs off between the time an agent ID is collected and the API is called.

Agent IDs are recycled, and are used to change the degree of parallelism for applications some time after being gathered by the database system monitor. When a user signs off, therefore, another user may sign on and acquire the same agent ID through this recycling process, with the result that the new degree of parallelism will be modified for the wrong user.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqlugrpn - Get the database partition server number for a row

Beginning with Version 9.7, this API is deprecated. Use the db2GetRowPartNum (Get the database partition server number for a row) API to return the database partition number and database partition server number for a row. If you call the sqlugrpn API and the DB2_PMAP_COMPATIBILITY registry variable is set to OFF, the error message SQL2768N is returned.

Returns the database partition number and the database partition server number based on the distribution key values. An application can use this information to determine on which database partition server a specific row of a table is stored.

The partitioning data structure, sqlupi, is the input for this API. The structure can be returned by the sqlugtpi API. Another input is the character representations of the corresponding distribution key values. The output is a database partition number generated by the distribution strategy and the corresponding database partition server number from the distribution map. If the distribution map information is not provided, only the database partition number is returned. This can be useful when analyzing data distribution.

The database manager does not need to be running when this API is called.

Scope

This API must be invoked from a database partition server in the db2nodes.cfg file. This API should not be invoked from a client, since it could result in erroneous database partitioning information being returned due to differences in codepage and endianess between the client and the server.

Authorization

None

API include file

`sqlutil.h`

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlugrpn (
   unsigned short num_ptrs,
   unsigned char ** ptr_array,
   unsigned short * ptr_lens,
   unsigned short territory_ctrycode,
   unsigned short codepage,
   struct sqlupi * part_info,
   short * part_num,
   SQL_PDB_NODE_TYPE * node_num,
```

```
   unsigned short chklvl,
   struct sqlca * sqlca,
   short dataformat,
   void * pReserved1,
   void * pReserved2);

SQL_API_RC SQL_API_FN
  sqlggrpn (
   unsigned short num_ptrs,
   unsigned char ** ptr_array,
   unsigned short * ptr_lens,
   unsigned short territory_code,
   unsigned short codepage,
   struct sqlupi * part_info,
   short * part_num,
   SQL_PDB_NODE_TYPE * node_num,
   unsigned short chklvl,
   struct sqlca * sqlca,
   short dataformat,
   void * pReserved1,
   void * pReserved2);
```

sqlugrpn API parameters

num_ptrs
The number of pointers in ptr_array. The value must be the same as the one specified for the part_info parameter; that is, part_info->sqld.

ptr_array
An array of pointers that points to the character representations of the corresponding values of each part of the distribution key specified in part_info. If a null value is required, the corresponding pointer is set to null. For generated columns, this function does not generate values for the row. The user is responsible for providing a value that will lead to the correct partitioning of the row.

ptr_lens
An array of unsigned integers that contains the lengths of the character representations of the corresponding values of each part of the partitioning key specified in part_info.

territory_ctrycode
The country/region code of the target database. This value can also be obtained from the database configuration file using the GET DATABASE CONFIGURATION command.

codepage
The code page of the target database. This value can also be obtained from the database configuration file using the GET DATABASE CONFIGURATION command.

part_info
A pointer to the sqlupi structure.

part_num
A pointer to a 2-byte signed integer that is used to store the database partition number.

node_num
A pointer to an SQL_PDB_NODE_TYPE field used to store the node number. If the pointer is null, no node number is returned.

chklvl An unsigned integer that specifies the level of checking that is done on

input parameters. If the value specified is zero, no checking is done. If any non-zero value is specified, all input parameters are checked.

sqlca Output. A pointer to the sqlca structure.

dataformat
Specifies the representation of distribution key values. Valid values are:

SQL_CHARSTRING_FORMAT
All distribution key values are represented by character strings. This is the default value.

SQL_IMPLIEDDECIMAL_FORMAT
The location of an implied decimal point is determined by the column definition. For example, if the column definition is DECIMAL(8,2), the value 12345 is processed as 123.45.

SQL_PACKEDDECIMAL_FORMAT
All decimal column distribution key values are in packed decimal format.

SQL_BINARYNUMERICS_FORMAT
All numeric distribution key values are in big-endian binary format.

pReserved1
Reserved for future use.

pReserved2
Reserved for future use.

Usage notes

Data types supported on the operating system are the same as those that can be defined as a distribution key.

Note: CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types must be converted to the database code page before this API is called.

For numeric and datetime data types, the character representations must be at the code page of the respective system where the API is invoked.

If node_num is not null, the distribution map must be supplied; that is, pmaplen field in part_info parameter (part_info->pmaplen) is either 2 or 8192. Otherwise, SQLCODE -6038 is returned. The distribution key must be defined; that is, sqld field in part_info parameter (part_info->sqld) must be greater than zero. Otherwise, SQLCODE -2032 is returned.

If a null value is assigned to a non-nullable partitioning column, SQLCODE -6039 is returned.

All the leading blanks and trailing blanks of the input character string are stripped, except for the CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types, where only trailing blanks are stripped.

sqlugtpi - Get table distribution information

Beginning with DB2 9.7, this API is deprecated. Use the db2GetDistMap (Get distribution map) API to return the distribution information. If you call the sqlugtpi API and the DB2_PMAP_COMPATIBILITY registry variable is set to OFF, the error message SQL2768N is returned.

Allows an application to obtain the distribution information for a table. The distribution information includes the distribution map and the column definitions of the distribution key. Information returned by this API can be passed to the sqlugrpn API to determine the database partition number and the database partition server number for any row in the table.

To use this API, the application must be connected to the database that contains the table for which distribution information is being requested.

Scope

This API can be executed on any database partition server defined in the `db2nodes.cfg` file.

Authorization

For the table being referenced, a user must have at least one of the following:

- DATAACCESS authority
- CONTROL privilege
- SELECT privilege

Required connection

Database

API include file

```
sqlutil.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlugtpi (
   unsigned char * tablename,
   struct sqlupi * part_info,
   struct sqlca * sqlca);

SQL_API_RC SQL_API_FN
  sqlggtpi (
   unsigned short tn_length,
   unsigned char * tablename,
   struct sqlupi * part_info,
   struct sqlca * sqlca);
```

sqlugtpi API parameters

tablename
The fully qualified name of the table.

part_info
A pointer to the sqlupi structure.

pSqlca
Output. A pointer to the sqlca structure.

sqlggtpi API-specific parameters

tn_length
A 2-byte unsigned integer with the length of the table name.

sqluvqdp - Quiesce table spaces for a table

Quiesces table spaces for a table. There are three valid quiesce modes: share, intent to update, and exclusive. There are three possible table space states resulting from the quiesce function:

- Quiesced: SHARE
- Quiesced: UPDATE
- Quiesced: EXCLUSIVE

Scope

In a single-partition database environment, this API quiesces all table spaces involved in a load operation in exclusive mode for the duration of the load. In a partitioned database environment, this API acts locally on a database partition. It quiesces only that portion of table spaces belonging to the database partition on which the load is performed.

Authorization

One of the following:

- sysadm
- sysctrl
- sysmaint
- dbadm
- load

Required connection

Database

API include file

```
sqlutil.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqluvqdp (
   char * pTableName,
   sqlint32 QuiesceMode,
   void * pReserved,
   struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgvqdp (
   unsigned short TableNameLen,
   char * pTableName,
   sqlint32 QuiesceMode,
   void * pReserved,
   struct sqlca * pSqlca);
```

sqluvqdp API parameters

pTableName
Input. A string containing the table name as used in the system catalog. This may be a two-part name with the schema and the table name separated by a period (.). If the schema is not provided, the CURRENT SCHEMA will be used.

The table cannot be a system catalog table. This field is mandatory.

QuiesceMode
Input. Specifies the quiesce mode. Valid values (defined in sqlutil) are:

SQLU_QUIESCEMODE_SHARE
For share mode

SQLU_QUIESCEMODE_INTENT_UPDATE
For intent to update mode

SQLU_QUIESCEMODE_EXCLUSIVE
For exclusive mode

SQLU_QUIESCEMODE_RESET
To reset the state of the table spaces to normal if either of the following is true:

- The caller owns the quiesce
- The caller who sets the quiesce disconnects, creating a ″phantom quiesce″

SQLU_QUIESCEMODE_RESET_OWNED
To reset the state of the table spaces to normal if the caller owns the quiesce.

This field is mandatory.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the sqlca structure.

sqlgvqdp API-specific parameters

TableNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the table name.

Usage notes

This API is not supported for declared temporary tables.

When the quiesce share request is received, the transaction requests intent share locks for the table spaces and a share lock for the table. When the transaction obtains the locks, the state of the table spaces is changed to QUIESCED SHARE. The state is granted to the quiescer only if there is no conflicting state held by other users. The state of the table spaces is recorded in the table space table, along with the authorization ID and the database agent ID of the quiescer, so that the state is persistent.

The table cannot be changed while the table spaces for the table are in QUIESCED SHARE state. Other share mode requests to the table and table spaces will be

allowed. When the transaction commits or rolls back, the locks are released, but the table spaces for the table remain in QUIESCED SHARE state until the state is explicitly reset.

When the quiesce exclusive request is made, the transaction requests super exclusive locks on the table spaces, and a super exclusive lock on the table. When the transaction obtains the locks, the state of the table spaces changes to QUIESCED EXCLUSIVE. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table. Since the table spaces are held in super exclusive mode, no other access to the table spaces is allowed. The user who invokes the quiesce function (the quiescer), however, has exclusive access to the table and the table spaces.

When a quiesce update request is made, the table spaces are locked in intent exclusive (IX) mode, and the table is locked in update (U) mode. The state of the table spaces with the quiescer is recorded in the table space table.

There is a limit of five quiescers on a table space at any given time. Since QUIESCED EXCLUSIVE is incompatible with any other state, and QUIESCED UPDATE is incompatible with another QUIESCED UPDATE, the five quiescer limit, if reached, must have at least four QUIESCED SHARE and at most one QUIESCED UPDATE.

A quiescer can upgrade the state of a table space from a less restrictive state to a more restrictive one (for example, S to U, or U to X). If a user requests a state lower than one that is already held, the original state is returned. States are not downgraded.

The quiesced state of a table space must be reset explicitly by using SQLU_QUIESCEMODE_RESET.

REXX API syntax

```
QUIESCE TABLESPACES FOR TABLE table_name
{SHARE | INTENT TO UPDATE | EXCLUSIVE | RESET}
```

REXX API parameters

table_name
Name of the table as used in the system catalog. This may be a two-part name with the schema and the table name separated by a period (.). If the schema is not provided, the CURRENT SCHEMA will be used.

DB2 APIs for Users

Following are the application programming interfaces (APIs) that correspond to the DB2 commands that are used for the Common Criteria evaluation.

sqlaprep - Precompile application program

Processes an application program source file containing embedded SQL statements. A modified source file is produced containing host language calls for the SQL statements and, by default, a package is created in the database.

Scope

This API can be called from any database partition server in db2nodes.cfg. It updates the database catalogs on the catalog partition. Its effects are visible to all database partition servers.

Authorization

One of the following authorizations:

- *dbadm* authority
- If EXPLAIN ONLY is specified, EXPLAIN authority or an authority that implicitly includes EXPLAIN is sufficient.
- If SQLERROR CHECK or EXPLAIN ONLY is specified, either EXPLAIN or SQLADM authority is sufficient.
- If a package does not exist, BINDADD authority and:
  - If the schema name of the package does not exist, IMPLICIT_SCHEMA authority on the database.
  - If the schema name of the package does exist, CREATEIN privilege on the schema.
- If the package exists, one of the following privileges:
  - ALTERIN privilege on the schema
  - BIND privilege on the package

In addition, if capturing explain information using the EXPLAIN or the EXPLSNAP clause, one of the following authorizations is required:

- INSERT privilege on the explain tables
- DATAACCESS authority

The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements.

Required connection

Database

API include file

```
sql.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlaprep (
        _SQLOLDCHAR * pProgramName,
        _SQLOLDCHAR * pMsgFileName,
        struct sqlopt * pPrepOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgprep (
        unsigned short MsgFileNameLen,
        unsigned short ProgramNameLen,
        struct sqlca * pSqlca,
        struct sqlopt * pPrepOptions,
        _SQLOLDCHAR * pMsgFileName,
        _SQLOLDCHAR * pProgramName);
```

sqlaprep API parameters

pProgramName
Input. A string containing the name of the application to be precompiled. Use the following extensions:

- .sqb: for COBOL applications
- .sqc: for C applications
- .sqC: for UNIX C++ applications
- .sqf: for FORTRAN applications
- .sqx: for C++ applications

When the TARGET option is used, the input file name extension does not have to be from this predefined list.

The preferred extension for C++ applications containing embedded SQL on UNIX based systems is sqC; however, the sqx convention, which was invented for systems that are not case sensitive, is tolerated by UNIX based systems.

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages. Can be the path and the name of an operating system file, or a standard device. If a file already exists, it is overwritten. If it does not exist, a file is created.

pPrepOptions
Input. A structure used to pass precompile options to the API. For more information about this structure, see SQLOPT.

pSqlca
Output. A pointer to the sqlca structure.

sqlgprep API-specific parameters

MsgFileNameLen
Input. Length in bytes of the pMsgFileName parameter.

ProgramNameLen
Input. Length in bytes of the pProgramName parameter.

Usage notes

A modified source file is produced, which contains host language equivalents to the SQL statements. By default, a package is created in the database to which a connection has been established. The name of the package is the same as the program file name (minus the extension and folded to uppercase), up to a maximum of 8 characters.

Following connection to a database, sqlaprep executes under the transaction that was started. PRECOMPILE PROGRAM then issues a COMMIT or a ROLLBACK operation to terminate the current transaction and start another one.

Precompiling stops if a fatal error or more than 100 errors occur. If a fatal error does occur, PRECOMPILE PROGRAM stops precompiling, attempts to close all files, and discards the package.

The Precompile option types and values are defined in sql.h.

When using the PRECOMPILE command or sqlaprep API, the name of the package can be specified with the PACKAGE USING option. When using this option, up to 128 bytes may be specified for the package name. When this option is not used, the name of the package is generated by the precompiler. The name of the application program source file (minus extension and folded to uppercase) is used up to a maximum of 8 characters. The name generated will continue to have a maximum of 8 bytes to be compatible with previous versions of DB2.

REXX API syntax

```
This API can be called from REXX through the SQLDB2 interface.
```

sqlarbnd - Rebind package

Allows the user to recreate a package stored in the database without the need for a bind file.

Authorization

One of the following:

- dbadm authority
- ALTERIN privilege on the schema
- BIND privilege on the package.

The authorization ID logged in the BOUNDBY column of the SYSCAT.PACKAGES system catalog table, which is the ID of the most recent binder of the package, is used as the binder authorization ID for the rebind, and for the default schema for table references in the package. Note that this default qualifier may be different from the authorization ID of the user executing the rebind request. REBIND will use the same bind options that were specified when the package was created.

Required connection

Database

API include file

```
sql.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlarbnd (
        char * pPackageName,
        struct sqlca * pSqlca,
        struct sqlopt * pRebindOptions);

SQL_API_RC SQL_API_FN
  sqlgrbnd (
        unsigned short PackageNameLen,
        char * pPackageName,
        struct sqlca * pSqlca,
        struct sqlopt * pRebindOptions);
```

sqlarbnd API parameters

pPackageName
Input. A string containing the qualified or unqualified name that designates the package to be rebound. An unqualified package-name is implicitly qualified by the current authorization ID. This name does not

include the package version. When specifying a package that has a version that is not the empty string, then the version-id must be specified using the SQL_VERSION_OPT rebind option.

pSqlca
Output. A pointer to the sqlca structure.

pRebindOptions
Input. A pointer to the SQLOPT structure, used to pass rebind options to the API. For more information about this structure, see SQLOPT.

sqlgrbnd API-specific parameters

PackageNameLen
Input. Length in bytes of the pPackageName parameter.

Usage notes

REBIND does not automatically commit the transaction following a successful rebind. The user must explicitly commit the transaction. This enables ″what if ″ analysis, in which the user updates certain statistics, and then tries to rebind the package to see what changes. It also permits multiple rebinds within a unit of work.

This API:

- Provides a quick way to recreate a package. This enables the user to take advantage of a change in the system without a need for the original bind file.fs. For example, if it is likely that a particular SQL statement can take advantage of a newly created index, REBIND can be used to recreate the package. REBIND can also be used to recreate packages after db2Runstats has been executed, thereby taking advantage of the new statistics.
- Provides a method to recreate inoperative packages. Inoperative packages must be explicitly rebound by invoking either the bind utility or the rebind utility. A package will be marked inoperative (the VALID column of the SYSCAT.PACKAGES system catalog will be set to X) if a function instance on which the package depends is dropped. The rebind conservative option is not supported for inoperative packages.
- Gives users control over the rebinding of invalid packages. Invalid packages will be automatically (or implicitly) rebound by the database manager when they are executed. This may result in a noticeable delay in the execution of the first SQL request for the invalid package. It may be desirable to explicitly rebind invalid packages, rather than allow the system to automatically rebind them, in order to eliminate the initial delay and to prevent unexpected SQL error messages which may be returned in case the implicit rebind fails. For example, following database upgrade, all packages stored in the database will be invalidated by the UPGRADE DATABASE command. Given that this may involve a large number of packages, it may be desirable to explicitly rebind all of the invalid packages at one time. This explicit rebinding can be accomplished using BIND, REBIND, or the db2rbind tool.

The choice of whether to use BIND or REBIND to explicitly rebind a package depends on the circumstances. It is recommended that REBIND be used whenever the situation does not specifically require the use of BIND, since the performance of REBIND is significantly better than that of BIND. BIND must be used, however:

- When there have been modifications to the program (for example, when SQL statements have been added or deleted, or when the package does not match the executable for the program).
- When the user wishes to modify any of the bind options as part of the rebind. REBIND does not support any bind options. For example, if the user wishes to have privileges on the package granted as part of the bind process, BIND must be used, since it has an SQL_GRANT_OPT option.
- When the package does not currently exist in the database.
- When detection of all bind errors is desired. REBIND only returns the first error it detects, and then ends, whereas the BIND command returns the first 100 errors that occur during binding.

REBIND is supported by DB2 Connect.

If REBIND is executed on a package that is in use by another user, the rebind will not occur until the other user's logical unit of work ends, because an exclusive lock is held on the package's record in the SYSCAT.PACKAGES system catalog table during the rebind.

When REBIND is executed, the database manager recreates the package from the SQL statements stored in the SYSCAT.STATEMENTS system catalog table. If many versions with the same package number and creator exist, only one version can be bound at once. If not specified using the SQL_VERSION_OPT rebind option, the VERSION defaults to be "". Even if there is only one package with a name and creator that matches the name and creator specified in the rebind request, it will not rebound unless its VERSION matches the VERSION specified explicitly or implicitly.

If REBIND encounters an error, processing stops, and an error message is returned.

The Explain tables are populated during REBIND if either SQL_EXPLSNAP_OPT or SQL_EXPLAIN_OPT have been set to YES or ALL (check EXPLAIN_SNAPSHOT and EXPLAIN_MODE columns in the catalog). The Explain tables used are those of the REBIND requester, not the original binder. The Rebind option types and values are defined in sql.h.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqleatcp - Attach to instance and change password

Enables an application to specify the node at which instance-level functions (CREATE DATABASE and FORCE APPLICATION, for example) are to be executed. This node may be the current instance (as defined by the value of the DB2INSTANCE environment variable), another instance on the same workstation, or an instance on a remote workstation. Establishes a logical instance attachment to the node specified, and starts a physical communications connection to the node if one does not already exist.

Note: This API extends the function of the sqleatin API by permitting the optional change of the user password for the instance being attached. The DB2 database system provides support for changing passwords on AIX, Linux and Windows operating systems.

Authorization

None

Required connection

This API establishes an instance attachment.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqleatcp (
        char * pNodeName,
        char * pUserName,
        char * pPassword,
        char * pNewPassword,
        struct sqlca * pSqlca);
```

sqleatcp API parameters

pNodeName
Input. A string containing the alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception is the local instance (as specified by the DB2INSTANCE environment variable), which can be specified as the object of an attachment, but cannot be used as a node name in the node directory. May be NULL.

pUserName
Input. A string containing the user name under which the attachment is to be authenticated. May be NULL.

pPassword
Input. A string containing the password for the specified user name. May be NULL.

pNewPassword
Input. A string containing the new password for the specified user name. Set to NULL if a password change is not required.

pSqlca
Output. A pointer to the sqlca structure.

Usage notes

A node name in the node directory can be regarded as an alias for an instance.

If an attach request succeeds, the sqlerrmc field of the sqlca will contain 9 tokens separated by hexadecimal FF (similar to the tokens returned when a CONNECT request is successful):

1. Country/region code of the application server
2. Code page of the application server
3. Authorization ID
4. Node name (as specified on the API)
5. Identity and platform type of the server
6. Agent ID of the agent which has been started at the server

7. Agent index
8. Node number of the server
9. Number of database partitions if the server is a partitioned database server.

If the node name is a zero-length string or NULL, information about the current state of attachment is returned. If no attachment exists, sqlcode 1427 is returned. Otherwise, information about the attachment is returned in the sqlerrmc field of the sqlca (as outlined above).

If an attachment has not been made, instance-level APIs are executed against the current instance, specified by the DB2INSTANCE environment variable.

Certain functions (db2start, db2stop, and all directory services, for example) are never executed remotely. That is, they affect only the local instance environment, as defined by the value of the DB2INSTANCE environment variable.

If an attachment exists, and the API is issued with a node name, the current attachment is dropped, and an attachment to the new node is attempted.

Where the user name and password are authenticated, and where the password is changed, depend on the authentication type of the target instance.

The node to which an attachment is to be made can also be specified by a call to the sqlesetc API.

REXX API syntax

Calling this API directly from REXX is not supported. However, REXX programmers can utilize this function by calling the DB2 command line processor to execute the ATTACH command.

sqleatin - Attach to instance

Enables an application to specify the node at which instance-level functions (CREATE DATABASE and FORCE APPLICATION, for example) are to be executed. This node may be the current instance (as defined by the value of the DB2INSTANCE environment variable), another instance on the same workstation, or an instance on a remote workstation. Establishes a logical instance attachment to the node specified, and starts a physical communications connection to the node if one does not already exist.

Note: If a password change is required, use the sqleatcp API instead of the sqleatin API.

Authorization

None

Required connection

This API establishes an instance attachment.

API include file

`sqlenv.h`

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqleatin (
        char * pNodeName,
        char * pUserName,
        char * pPassword,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgatin (
        unsigned short PasswordLen,
        unsigned short UserNameLen,
        unsigned short NodeNameLen,
        struct sqlca * pSqlca,
        char * pPassword,
        char * pUserName,
        char * pNodeName);
```

sqleatin API parameters

pNodeName
Input. A string containing the alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception is the local instance (as specified by the DB2INSTANCE environment variable), which can be specified as the object of an attachment, but cannot be used as a node name in the node directory. Can be NULL.

pUserName
Input. A string containing the user name under which the attachment is to be authenticated. Can be NULL.

pPassword
Input. A string containing the password for the specified user name. Can be NULL.

pSqlca
Output. A pointer to the sqlca structure.

sqlgatin API-specific parameters

PasswordLen
Input. A 2-byte unsigned integer representing the length of the password in bytes. Set to zero if no password is supplied.

UserNameLen
Input. A 2-byte unsigned integer representing the length of the user name in bytes. Set to zero if no user name is supplied.

NodeNameLen
Input. A 2-byte unsigned integer representing the length of the node name in bytes. Set to zero if no node name is supplied.

Usage notes

Note: A node name in the node directory can be regarded as an alias for an instance.

If an attach request succeeds, the sqlerrmc field of the sqlca will contain 9 tokens separated by hexadecimal FF (similar to the tokens returned when a CONNECT request is successful):

1. Country/region code of the application server

2. Code page of the application server
3. Authorization ID
4. Node name (as specified on the API)
5. Identity and platform type of the server
6. Agent ID of the agent which has been started at the server
7. Agent index
8. Node number of the server
9. Number of database partitions if the server is a partitioned database server.

If the node name is a zero-length string or NULL, information about the current state of attachment is returned. If no attachment exists, sqlcode 1427 is returned. Otherwise, information about the attachment is returned in the sqlerrmc field of the sqlca (as outlined above).

If an attachment has not been made, instance-level APIs are executed against the current instance, specified by the DB2INSTANCE environment variable.

Certain functions (db2start, db2stop, and all directory services, for example) are never executed remotely. That is, they affect only the local instance environment, as defined by the value of the DB2INSTANCE environment variable.

If an attachment exists, and the API is issued with a node name, the current attachment is dropped, and an attachment to the new node is attempted.

Where the user name and password are authenticated depends on the authentication type of the target instance.

The node to which an attachment is to be made can also be specified by a call to the sqlesetc API.

REXX API syntax

```
ATTACH [TO nodename [USER username USING password]]
```

REXX API parameters

nodename
Alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception is the local instance (as specified by the DB2INSTANCE environment variable), which can be specified as the object of an attachment, but cannot be used as a node name in the node directory.

username
Name under which the user attaches to the instance.

password
Password used to authenticate the user name.

sqledtin - Detach from instance

Removes the logical instance attachment, and terminates the physical communication connection if there are no other logical connections using this layer.

Authorization

None

Required connection

None. Removes an existing instance attachment.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqledtin (
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdtin (
        struct sqlca * pSqlca);
```

sqledtin API parameters

pSqlca
Output. A pointer to the sqlca structure.

REXX API syntax

```
DETACH
```

Part 4. Appendixes

Appendix A. Related topics (linked to from topics in this book)

SQL Reference topics

SYSCAT.ATTRIBUTES

Each row represents an attribute that is defined for a user-defined structured data type. Includes inherited attributes of subtypes.

Table 172. SYSCAT.ATTRIBUTES Catalog View

Column Name	Data Type	Nullable	Description
TYPESCHEMA	VARCHAR (128)		Schema name of the structured data type that includes the attribute.
TYPEMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the structured data type belongs. The null value if not a module structured data type.
TYPENAME	VARCHAR (128)		Unqualified name of the structured data type that includes the attribute.
ATTR_NAME	VARCHAR (128)		Attribute name.
ATTR_TYPESCHEMA	VARCHAR (128)		Schema name of the data type of an attribute.
ATTR_TYPEMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the data type of an attribute belongs. The null value if not a module attribute.
ATTR_TYPENAME	VARCHAR (128)		Unqualified name of the data type of an attribute.
TARGET_TYPESCHEMA	VARCHAR (128)	Y	Schema name of the target row type. Applies to reference types only; null value otherwise.
TARGET_TYPEMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the target row type belongs. The null value if not a module row type. Applies to reference types only; null value otherwise.
TARGET_TYPENAME	VARCHAR (128)	Y	Unqualified name of the target row type. Applies to reference types only; null value otherwise.
SOURCE_TYPESCHEMA	VARCHAR (128)		For inherited attributes, the schema name of the data type with which the attribute was first defined. For non-inherited attributes, this column is the same as TYPESCHEMA.
SOURCE_TYPEMODULENAME	VARCHAR(128)	Y	For inherited attributes, the unqualified name of the module to which the data type with which the attribute was first defined belongs. For non-inherited attributes, this column is the same as TYPEMODULEID. The null value if not a module data type.

Table 172. SYSCAT.ATTRIBUTES Catalog View (continued)

Column Name	Data Type	Nullable	Description
SOURCE_TYPENAME	VARCHAR (128)		For inherited attributes, the unqualified name of the data type with which the attribute was first defined. For non-inherited attributes, this column is the same as TYPENAME.
ORDINAL	SMALLINT		Position of the attribute in the definition of the structured data type, starting with 0.
LENGTH	INTEGER		Length of the attribute data type. 0 if the attribute is a user-defined type.
SCALE	SMALLINT		Scale if the attribute data type is DECIMAL or distinct type based on DECIMAL; the number of digits of fractional seconds if the attribute data type is TIMESTAMP or distinct type based on TIMESTAMP; 0 otherwise.
CODEPAGE	SMALLINT		For string types, denotes the code page; 0 indicates FOR BIT DATA; 0 for non-string types.
COLLATIONSCHEMA	VARCHAR (128)	Y	For string types, the schema name of the collation for the attribute; the null value otherwise.
COLLATIONNAME	VARCHAR (128)	Y	For string types, the unqualified name of the collation for the attribute; the null value otherwise.
LOGGED	CHAR (1)		Applies to LOB types only; blank otherwise. • N = Changes are not logged • Y = Changes are logged
COMPACT	CHAR (1)		Applies to LOB types only; blank otherwise. • N = Stored in non-compact format • Y = Stored in compact format
DL_FEATURES	CHAR(10)		This column is no longer used and will be removed in a future release.
JAVA_FIELDNAME	VARCHAR (256)	Y	Reserved for future use.

SYSCAT.AUDITPOLICIES

Each row represents an audit policy.

Table 173. SYSCAT.AUDITPOLICIES Catalog View

Column Name	Data Type	Nullable	Description
AUDITPOLICYNAME	VARCHAR (128)		Name of the audit policy.
AUDITPOLICYID	INTEGER		Identifier for the audit policy.
CREATE_TIME	TIMESTAMP		Time at which the audit policy was created.
ALTER_TIME	TIMESTAMP		Time at which the audit policy was last altered.

Table 173. SYSCAT.AUDITPOLICIES Catalog View (continued)

Column Name	Data Type	Nullable	Description
AUDITSTATUS	CHAR (1)		Status for the AUDIT category. • B = Both • F = Failure • N = None • S = Success
CONTEXTSTATUS	CHAR (1)		Status for the CONTEXT category. • B = Both • F = Failure • N = None • S = Success
VALIDATESTATUS	CHAR (1)		Status for the VALIDATE category. • B = Both • F = Failure • N = None • S = Success
CHECKINGSTATUS	CHAR (1)		Status for the CHECKING category. • B = Both • F = Failure • N = None • S = Success
SECMAINTSTATUS	CHAR (1)		Status for the SECMAINT category. • B = Both • F = Failure • N = None • S = Success
OBJMAINTSTATUS	CHAR (1)		Status for the OBJMAINT category. • B = Both • F = Failure • N = None • S = Success
SYSADMINSTATUS	CHAR (1)		Status for the SYSADMIN category. • B = Both • F = Failure • N = None • S = Success
EXECUTESTATUS	CHAR (1)		Status for the EXECUTE category. • B = Both • F = Failure • N = None • S = Success
EXECUTEWITHDATA	CHAR (1)		Host variables and parameter markers logged with EXECUTE category. • N = No • Y = Yes

Table 173. SYSCAT.AUDITPOLICIES Catalog View (continued)

Column Name	Data Type	Nullable	Description
ERRORTYPE	CHAR (1)		The audit error type. • A = Audit • N = Normal
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.AUDITUSE

Each row represents an audit policy that is associated with a non-database object, such as USER, GROUP, or authority (SYSADM, SYSCTRL, SYSMAINT).

Table 174. SYSCAT.AUDITUSE Catalog View

Column Name	Data Type	Nullable	Description
AUDITPOLICYNAME	VARCHAR (128)		Name of the audit policy.
AUDITPOLICYID	INTEGER		Identifier for the audit policy.
OBJECTTYPE	CHAR(1)		The type of object with which this audit policy is associated. • S = MQT • T = Table • g = Authority • i = Authorization ID • x = Trusted context • Blank = Database
SUBOBJECTTYPE	CHAR(1)		If OBJECTTYPE is 'i', this is the type that the authorization ID represents. • G = Group • R = Role • U = User • Blank = Not applicable
OBJECTSCHEMA	VARCHAR (128)		Schema name of the object for which the audit policy is in use. OBJECTSCHEMA is null if OBJECTTYPE identifies an object to which a schema does not apply.
OBJECTNAME	VARCHAR (128)		Unqualified name of the object for which this audit policy is in use.

SYSCAT.BUFFERPOOLDBPARTITIONS

Each row represents a combination of a buffer pool and a database partition, in which the size of the buffer pool on that partition is different from its default size for other partitions in the same database partition group (as represented in SYSCAT.BUFFERPOOLS).

Table 175. SYSCAT.BUFFERPOOLDBPARTITIONS Catalog View

Column Name	Data Type	Nullable	Description
BUFFERPOOLID	INTEGER		Internal buffer pool identifier.
DBPARTITIONNUM	SMALLINT		Database partition number.

Table 175. SYSCAT.BUFFERPOOLDBPARTITIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
NPAGES	INTEGER		Number of pages in this buffer pool on this database partition.

SYSCAT.BUFFERPOOLS

Each row represents the configuration of a buffer pool on one database partition group of a database, or on all database partitions of a database.

Table 176. SYSCAT.BUFFERPOOLS Catalog View

Column Name	Data Type	Nullable	Description
BPNAME	VARCHAR (128)		Name of the buffer pool.
BUFFERPOOLID	INTEGER		Identifier for the buffer pool.
DBPGNAME	VARCHAR (128)	Y	Name of the database partition group (the null value if the buffer pool exists on all database partitions in the database).
NPAGES	INTEGER		Default number of pages in this buffer pool on database partitions in this database partition group.
PAGESIZE	INTEGER		Page size for this buffer pool on database partitions in this database partition group.
ESTORE	INTEGER		Always 'N'. Extended storage no longer applies.
NUMBLOCKPAGES	INTEGER		Number of pages of the buffer pool that are to be in a block-based area. A block-based area of the buffer pool is only used by prefetchers doing a sequential prefetch.
BLOCKSIZE	INTEGER		Number of pages in a *block*.
NGNAME[1]	VARCHAR (128)	Y	Name of the database partition group (the null value if the buffer pool exists on all database partitions in the database).

Note:

1. The NGNAME column is included for backwards compatibility. See DBPGNAME.

SYSCAT.CASTFUNCTIONS

Each row represents a cast function, not including built-in cast functions.

Table 177. SYSCAT.CASTFUNCTIONS Catalog View

Column Name	Data Type	Nullable	Description
FROM_TYPESCHEMA	VARCHAR (128)		Schema name of the data type of the parameter.
FROM_TYPEMODULENAME	VARCHAR (128)		Unqualified name of the module to which the data type of the parameter belongs. The null value if not a module data type.
FROM_TYPENAME	VARCHAR (128)		Name of the data type of the parameter.

Table 177. SYSCAT.CASTFUNCTIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
FROM_TYPEMODULEID	INTEGER	Y	Identifier for the module to which the data type of the parameter belongs. The null value if not a module data type.
TO_TYPESCHEMA	VARCHAR (128)		Schema name of the data type of the result after casting.
TO_TYPEMODULENAME	VARCHAR (128)		Unqualified name of the module to which the data type of the result after casting belongs. The null value if not a module data type.
TO_TYPENAME	VARCHAR (128)		Name of the data type of the result after casting.
TO_TYPEMODULEID	INTEGER	Y	Identifier for the module to which the data type of the result after casting belongs. The null value if not a module data type.
FUNCSCHEMA	VARCHAR (128)		Schema name of the function.
FUNCMODULENAME	VARCHAR (128)		Unqualified name of the module to which the function belongs. The null value if not a module function.
FUNCNAME	VARCHAR (128)		Unqualified name of the function.
SPECIFICNAME	VARCHAR (128)		Name of the routine instance (might be system-generated).
FUNCMODULEID	INTEGER	Y	Identifier for the module to which the function belongs. The null value if not a module function.
ASSIGN_FUNCTION	CHAR (1)		• N = Not an assignment function • Y = Implicit assignment function

SYSCAT.CHECKS

Each row represents a check constraint or a derived column in a materialized query table. For table hierarchies, each check constraint is recorded only at the level of the hierarchy where the constraint was created.

Table 178. SYSCAT.CHECKS Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR (128)		Name of the check constraint.
OWNER	VARCHAR (128)		Authorization ID of the owner of the constraint.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
TABSCHEMA	VARCHAR (128)		Schema name of the table to which this constraint applies.
TABNAME	VARCHAR (128)		Name of the table to which this constraint applies.
CREATE_TIME	TIMESTAMP		Time at which the constraint was defined. Used in resolving functions that are part of this constraint. Functions that were created after the constraint was defined are not chosen.
QUALIFIER	VARCHAR (128)		Value of the default schema at the time of object definition. Used to complete any unqualified references.

Table 178. SYSCAT.CHECKS Catalog View (continued)

Column Name	Data Type	Nullable	Description
TYPE	CHAR (1)		Type of check constraint: • C = Check constraint • F = Functional dependency • O = Constraint is an object property • S = System-generated check constraint for a GENERATED ALWAYS column
FUNC_PATH	CLOB (2K)		SQL path in effect when the constraint was defined.
TEXT	CLOB (2M)		Text of the check condition or definition of the derived column.[1]
PERCENTVALID	SMALLINT		Number of rows for which the informational constraint is valid, expressed as a percentage of the total.
COLLATIONSCHEMA	VARCHAR (128)		Schema name of the collation for the constraint.
COLLATIONNAME	VARCHAR (128)		Unqualified name of the collation for the constraint.
COLLATIONSCHEMA_ORDERBY	VARCHAR (128)		Schema name of the collation for ORDER BY clauses in the constraint.
COLLATIONNAME_ORDERBY	VARCHAR (128)		Unqualified name of the collation for ORDER BY clauses in the constraint.
DEFINER[2]	VARCHAR (128)		Authorization ID of the owner of the constraint.

Note:

1. In the catalog view, the text of the check condition is always shown in the database code page and can contain substitution characters. The check constraint will always be applied in the code page of the target table, and will not contain any substitution characters when applied. (The check constraint will be applied based on the original text in the code page of the target table, which might not include the substitution characters.)
2. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.COLCHECKS

Each row represents a column that is referenced by a check constraint or by the definition of a materialized query table. For table hierarchies, each check constraint is recorded only at the level of the hierarchy where the constraint was created.

Table 179. SYSCAT.COLCHECKS Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR (128)		Name of the check constraint.
TABSCHEMA	VARCHAR (128)		Schema name of the table containing the referenced column.
TABNAME	VARCHAR (128)		Unqualified name of the table containing the referenced column.
COLNAME	VARCHAR (128)		Name of the column.

Table 179. SYSCAT.COLCHECKS Catalog View (continued)

Column Name	Data Type	Nullable	Description
USAGE	CHAR (1)		• D = Column is the child in a functional dependency • P = Column is the parent in a functional dependency • R = Column is referenced in the check constraint • S = Column is a source in the system-generated column check constraint that supports a materialized query table • T = Column is a target in the system-generated column check constraint that supports a materialized query table

SYSCAT.COLDIST

Each row represents the *n*th most frequent value of some column, or the *n*th quantile (cumulative distribution) value of the column. Applies to columns of real tables only (not views). No statistics are recorded for inherited columns of typed tables.

Table 180. SYSCAT.COLDIST Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the table to which the statistics apply.
TABNAME	VARCHAR (128)		Unqualified name of the table to which the statistics apply.
COLNAME	VARCHAR (128)		Name of the column to which the statistics apply.
TYPE	CHAR (1)		• F = Frequency value • Q = Quantile value
SEQNO	SMALLINT		If TYPE = 'F', *n* in this column identifies the *n*th most frequent value. If TYPE = 'Q', *n* in this column identifies the *n*th quantile value.
COLVALUE[1]	VARCHAR (254)	Y	Data value as a character literal or a null value.
VALCOUNT	BIGINT		If TYPE = 'F', VALCOUNT is the number of occurrences of COLVALUE in the column. If TYPE = 'Q', VALCOUNT is the number of rows whose value is less than or equal to COLVALUE.
DISTCOUNT[2]	BIGINT	Y	If TYPE = 'Q', this column records the number of distinct values that are less than or equal to COLVALUE (the null value if unavailable).

Note:

1. In the catalog view, the value of COLVALUE is always shown in the database code page and can contain substitution characters. However, the statistics are gathered internally in the code page of the column's table, and will therefore use actual column values when applied during query optimization.
2. DISTCOUNT is collected only for columns that are the first key column in an index.

SYSCAT.COLGROUPCOLS

Each row represents a column that makes up a column group.

Table 181. SYSCAT.COLGROUPCOLS Catalog View

Column Name	Data Type	Nullable	Description
COLGROUPID	INTEGER		Identifier for the column group.
COLNAME	VARCHAR (128)		Name of the column in the column group.
TABSCHEMA	VARCHAR (128)		Schema name of the table for the column in the column group.
TABNAME	VARCHAR (128)		Unqualified name of the table for the column in the column group.
ORDINAL	SMALLINT		Ordinal number of the column in the column group.

SYSCAT.COLGROUPDIST

Each row represents the value of the column in a column group that makes up the *n*th most frequent value of the column group or the *n*th quantile value of the column group.

Table 182. SYSCAT.COLGROUPDIST Catalog View

Column Name	Data Type	Nullable	Description
COLGROUPID	INTEGER		Identifier for the column group.
TYPE	CHAR (1)		• F = Frequency value • Q = Quantile value
ORDINAL	SMALLINT		Ordinal number of the column in the column group.
SEQNO	SMALLINT		If TYPE = 'F', *n* in this column identifies the *n*th most frequent value. If TYPE = 'Q', *n* in this column identifies the *n*th quantile value.
COLVALUE[1]	VARCHAR (254)		Data value as a character literal or a null value.

Note:

1. In the catalog view, the value of COLVALUE is always shown in the database code page and can contain substitution characters. However, the statistics are gathered internally in the code page of the column's table, and will therefore use actual column values when applied during query optimization.

SYSCAT.COLGROUPDISTCOUNTS

Each row represents the distribution statistics that apply to the *n*th most frequent value of a column group or the *n*th quantile of a column group.

Table 183. SYSCAT.COLGROUPDISTCOUNTS Catalog View

Column Name	Data Type	Nullable	Description
COLGROUPID	INTEGER		Identifier for the column group.

Table 183. SYSCAT.COLGROUPDISTCOUNTS Catalog View (continued)

Column Name	Data Type	Nullable	Description
TYPE	CHAR (1)		• F = Frequency value • Q = Quantile value
SEQNO	SMALLINT		Sequence number *n* representing the *n*th TYPE value.
VALCOUNT	BIGINT		If TYPE = 'F', VALCOUNT is the number of occurrences of COLVALUE for the column group with this SEQNO. If TYPE = 'Q', VALCOUNT is the number of rows whose value is less than or equal to COLVALUE for the column group with this SEQNO.
DISTCOUNT	BIGINT		If TYPE = 'Q', this column records the number of distinct values that are less than or equal to COLVALUE for the column group with this SEQNO (the null value if unavailable).

SYSCAT.COLGROUPS

Each row represents a column group and statistics that apply to the entire column group.

Table 184. SYSCAT.COLGROUPS Catalog View

Column Name	Data Type	Nullable	Description
COLGROUPSCHEMA	VARCHAR (128)		Schema name of the column group.
COLGROUPNAME	VARCHAR (128)		Unqualified name of the column group.
COLGROUPID	INTEGER		Identifier for the column group.
COLGROUPCARD	BIGINT		Cardinality of the column group.
NUMFREQ_VALUES	SMALLINT		Number of frequent values collected for the column group.
NUMQUANTILES	SMALLINT		Number of quantiles collected for the column group.

SYSCAT.COLIDENTATTRIBUTES

Each row represents an identity column that is defined for a table.

Table 185. SYSCAT.COLIDENTATTRIBUTES Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the table or view that contains the column.
TABNAME	VARCHAR (128)		Unqualified name of the table or view that contains the column.
COLNAME	VARCHAR (128)		Name of the column.
START	DECIMAL (31,0)	Y	Start value of the sequence. The null value if the sequence is an alias.
INCREMENT	DECIMAL (31,0)	Y	Increment value. The null value if the sequence is an alias.

Table 185. SYSCAT.COLIDENTATTRIBUTES Catalog View (continued)

Column Name	Data Type	Nullable	Description
MINVALUE	DECIMAL (31,0)	Y	Minimum value of the sequence. The null value if the sequence is an alias.
MAXVALUE	DECIMAL (31,0)	Y	Maximum value of the sequence. The null value if the sequence is an alias.
CYCLE	CHAR (1)		Indicates whether or not the sequence can continue to generate values after reaching its maximum or minimum value. • N = Sequence cannot cycle • Y = Sequence can cycle • Blank = Sequence is an alias.
CACHE	INTEGER		Number of sequence values to pre-allocate in memory for faster access. 0 indicates that values of the sequence are not to be preallocated. In a partitioned database, this value applies to each database partition. -1 if the sequence is an alias.
ORDER	CHAR (1)		Indicates whether or not the sequence numbers must be generated in order of request. • N = Sequence numbers are not required to be generated in order of request • Y = Sequence numbers must be generated in order of request • Blank = Sequence is an alias.
NEXTCACHEFIRSTVALUE	DECIMAL (31,0)	Y	The first value available to be assigned in the next cache block. If no caching, the next value available to be assigned.
SEQID	INTEGER		Identifier for the sequence or alias.

SYSCAT.COLOPTIONS

Each row contains column-specific option values.

Table 186. SYSCAT.COLOPTIONS Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the nickname.
TABNAME	VARCHAR (128)		Nickname for the column for which options are set.
COLNAME	VARCHAR (128)		Local column name.
OPTION	VARCHAR (128)		Name of the column option.
SETTING	CLOB (32K)		Value.

SYSCAT.COLUMNS

Each row represents a column defined for a table, view, or nickname.

Table 187. SYSCAT.COLUMNS Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the table, view, or nickname that contains the column.
TABNAME	VARCHAR (128)		Unqualified name of the table, view, or nickname that contains the column.
COLNAME	VARCHAR (128)		Name of the column.
COLNO	SMALLINT		Number of this column in the table (starting with 0).
TYPESCHEMA	VARCHAR (128)		Schema name of the data type for the column.
TYPENAME	VARCHAR (128)		Unqualified name of the data type for the column.
LENGTH	INTEGER		Maximum length of the data; 0 for distinct types. The LENGTH column indicates precision for DECIMAL fields, and indicates the number of bytes of storage required for decimal floating-point columns; that is, 8 and 16 for DECFLOAT(16) and DECFLOAT(34), respectively.
SCALE	SMALLINT		Scale if the column type is DECIMAL or number of digits of fractional seconds if the column type is TIMESTAMP; 0 otherwise.
DEFAULT[1]	VARCHAR (254)	Y	Default value for the column of a table expressed as a constant, special register, or cast-function appropriate for the data type of the column. Can also be the keyword NULL. Values might be converted from what was specified as a default value. For example, date and time constants are shown in ISO format, cast-function names are qualified with schema names, and identifiers are delimited. Null value if a DEFAULT clause was not specified or the column is a view column.
NULLS[2]	CHAR (1)		Nullability attribute for the column. • N = Column is not nullable • Y = Column is nullable The value can be 'N' for a view column that is derived from an expression or function. Nevertheless, such a column allows null values when the statement using the view is processed with warnings for arithmetic errors.
CODEPAGE	SMALLINT		Code page used for data in this column; 0 if the column is defined as FOR BIT DATA or is not a string type.
COLLATIONSCHEMA	VARCHAR (128)	Y	For string types, the schema name of the collation for the column; the null value otherwise.
COLLATIONNAME	VARCHAR (128)	Y	For string types, the unqualified name of the collation for the column; the null value otherwise.

Table 187. SYSCAT.COLUMNS Catalog View (continued)

Column Name	Data Type	Nullable	Description
LOGGED	CHAR (1)		Applies only to columns whose type is LOB or distinct based on LOB; blank otherwise. • N = Column is not logged • Y = Column is logged
COMPACT	CHAR (1)		Applies only to columns whose type is LOB or distinct based on LOB; blank otherwise. • N = Column is not compacted • Y = Column is compacted in storage
COLCARD	BIGINT		Number of distinct values in the column; -1 if statistics are not collected; -2 for inherited columns and columns of hierarchy tables.
HIGH2KEY[3]	VARCHAR (254)	Y	Second-highest data value. Representation of numeric data changed to character literals. Empty if statistics are not collected. Empty for inherited columns and columns of hierarchy tables.
LOW2KEY[3]	VARCHAR (254)	Y	Second-lowest data value. Representation of numeric data changed to character literals. Empty if statistics are not collected. Empty for inherited columns and columns of hierarchy tables.
AVGCOLLEN	INTEGER		Average space in bytes when the column is stored in database memory or a temporary table. For LOB data types that are not inlined, LONG data types, and XML documents, the value used to calculate the average column length is the length of the data descriptor. An extra byte is required if the column is nullable; -1 if statistics have not been collected; -2 for inherited columns and columns of hierarchy tables. Note: The average space required to store the column on disk may be different than the value represented by this statistic.
KEYSEQ	SMALLINT	Y	The column's numerical position within the table's primary key. The null value for columns of subtables and hierarchy tables.
PARTKEYSEQ	SMALLINT	Y	The column's numerical position within the table's distribution key; 0 or the null value if the column is not in the distribution key. The null value for columns of subtables and hierarchy tables.
NQUANTILES	SMALLINT		Number of quantile values recorded in SYSCAT.COLDIST for this column; -1 if statistics are not gathered; -2 for inherited columns and columns of hierarchy tables.
NMOSTFREQ	SMALLINT		Number of most-frequent values recorded in SYSCAT.COLDIST for this column; -1 if statistics are not gathered; -2 for inherited columns and columns of hierarchy tables.
NUMNULLS	BIGINT		Number of null values in the column; -1 if statistics are not collected.

Table 187. SYSCAT.COLUMNS Catalog View (continued)

Column Name	Data Type	Nullable	Description
TARGET_TYPESCHEMA	VARCHAR (128)	Y	Schema name of the target row type, if the type of this column is REFERENCE; null value otherwise.
TARGET_TYPENAME	VARCHAR (128)	Y	Unqualified name of the target row type, if the type of this column is REFERENCE; null value otherwise.
SCOPE_TABSCHEMA	VARCHAR (128)	Y	Schema name of the scope (target table), if the type of this column is REFERENCE; null value otherwise.
SCOPE_TABNAME	VARCHAR (128)	Y	Unqualified name of the scope (target table), if the type of this column is REFERENCE; null value otherwise.
SOURCE_TABSCHEMA	VARCHAR (128)	Y	For columns of typed tables or views, the schema name of the table or view in which the column was first introduced. For non-inherited columns, this is the same as TABSCHEMA. The null value for columns of non-typed tables and views.
SOURCE_TABNAME	VARCHAR (128)	Y	For columns of typed tables or views, the unqualified name of the table or view in which the column was first introduced. For non-inherited columns, this is the same as TABNAME. The null value for columns of non-typed tables and views.
DL_FEATURES	CHAR (10)	Y	This column is no longer used and will be removed in a future release.
SPECIAL_PROPS	CHAR (8)	Y	Applies to REFERENCE type columns only; blanks otherwise. Each byte position is defined as follows: • 1 = Object identifier (OID) column ('Y' for yes; 'N' for no) • 2 = User-generated or system-generated ('U' for user; 'S' for system) Bytes 3 through 8 are reserved for future use.
HIDDEN	CHAR (1)		Type of hidden column. • I = Column is defined as IMPLICITLY HIDDEN • S = System-managed hidden column • Blank = Column is not hidden
INLINE_LENGTH	INTEGER		Maximum size in bytes of the internal representation of an instance of an XML document, a structured type, or a LOB data type, that can be stored in the base table; 0 when not applicable.
PCTINLINED	SMALLINT		Percentage of inlined XML documents or LOB data. -1 if statistics have not been collected.

Table 187. SYSCAT.COLUMNS Catalog View (continued)

Column Name	Data Type	Nullable	Description
IDENTITY	CHAR (1)		• N = Not an identity column • T = Row change timestamp column • Y = Identity column
ROWCHANGETIMESTAMP	CHAR (1)		• N = Not a row change timestamp column • Y = Row change timestamp column
GENERATED	CHAR (1)		Type of generated column. • A = Column value is always generated • D = Column value is generated by default • Blank = Column is not generated
TEXT	CLOB (2M)	Y	For columns defined as generated as expression, this field contains the text of the generated column expression, starting with the keyword AS.
COMPRESS	CHAR (1)		• O = Compress off • S = Compress system default values
AVGDISTINCTPERPAGE	DOUBLE	Y	For future use.
PAGEVARIANCERATIO	DOUBLE	Y	For future use.
SUB_COUNT	SMALLINT		Average number of sub-elements in the column. Applicable to character string columns only.
SUB_DELIM_LENGTH	SMALLINT		Average length of the delimiters that separate each sub-element in the column. Applicable to character string columns only.
AVGCOLLENCHAR	INTEGER		Average number of characters (based on the collation in effect for the column) required for the column; -1 if the data type of the column is long, LOB, or XML or if statistics have not been collected; -2 for inherited columns and columns of hierarchy tables.
IMPLICITVALUE[4]	VARCHAR (254)	Y	For a column that was added to a table after the table was created, stores the default value at the time the column was added. For a column that was defined when the table was created, stores the null value.
SECLABELNAME	VARCHAR(128)	Y	Name of the security label that is associated with the column if it is a protected column; the null value otherwise.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Table 187. SYSCAT.COLUMNS Catalog View (continued)

Column Name	Data Type	Nullable	Description

Note:

1. For Version 2.1.0, cast-function names were not delimited and may still appear this way in the DEFAULT column. Also, some view columns included default values which will still appear in the DEFAULT column.
2. Starting with Version 2, value D (indicating not null with a default) is no longer used. Instead, use of WITH DEFAULT is indicated by a non-null value in the DEFAULT column.
3. In the catalog view, the values of HIGH2KEY and LOW2KEY are always shown in the database code page and can contain substitution characters. However, the statistics are gathered internally in the code page of the column's table, and will therefore use actual column values when applied during query optimization.
4. Attaching a data partition is allowed unless IMPLICITVALUE for a specific column is a non-null value for both the source column and the target column, and the values do not match. In this case, you must drop the source table and then recreate it. A column can have a non-null value in the IMPLICITVALUE field if one of the following conditions is met:
   - The column is created as the result of an ALTER TABLE...ADD COLUMN statement
   - The IMPLICITVALUE field is propagated from a source table during attach
   - The IMPLICITVALUE field is inherited from a source table during detach
   - The IMPLICITVALUE field is set during database upgrade from Version 8 to Version 9, where it is determined to be an added column, or might be an added column. If the database is not certain whether the column is added or not, it is treated as added. An added column is a column that was created as the result of an ALTER TABLE...ADD COLUMN statement.

   To avoid these inconsistencies during non-migration scenarios, it is recommended that you always create the tables that you are going to attach with all the columns already defined. That is, never use the ALTER TABLE statement to add columns to a table before attaching it.

SYSCAT.COLUSE

Each row represents a column that is referenced in the DIMENSIONS clause of a CREATE TABLE statement.

Table 188. SYSCAT.COLUSE Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the table containing the column.
TABNAME	VARCHAR (128)		Unqualified name of the table containing the column.
COLNAME	VARCHAR (128)		Name of the column.
DIMENSION	SMALLINT		Dimension number, based on the order of dimensions specified in the DIMENSIONS clause (initial position is 0). For a composite dimension, this value will be the same for each component of the dimension.
COLSEQ	SMALLINT		Numeric position of the column in the dimension to which it belongs (initial position is 0). The value is 0 for the single column in a noncomposite dimension.
TYPE	CHAR (1)		Type of dimension. • C = Clustering or multidimensional clustering • P = Partitioning

SYSCAT.CONDITIONS

Each row represents a condition defined in a module.

Table 189. SYSCAT.CONDITIONS Catalog View

Column Name	Data Type	Nullable	Description
CONDSCHEMA	VARCHAR (128)		Schema name of the condition.
CONDMODULENAME	VARCHAR (128)	Y	Unqualified name of the module to which the condition belongs.
CONDNAME	VARCHAR (128)		Unqualified name of the condition.
CONDID	INTEGER		Identifier for the condition.
CONDMODULEID	INTEGER	Y	Identifier of the module to which the condition belongs.
SQLSTATE	CHAR(5)	Y	SQLSTATE value associated with the condition.
OWNER	VARCHAR (128)		Authorization ID of the owner of the condition.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
CREATE_TIME	TIMESTAMP		Time at which the condition was created.
REMARKS	VARCHAR (254)		User-provided comments, or the null value.

SYSCAT.CONSTDEP

Each row represents a dependency of a constraint on some other object. The constraint depends on the object of type BTYPE of name BNAME, so a change to the object affects the constraint.

Table 190. SYSCAT.CONSTDEP Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR (128)		Unqualified name of the constraint.
TABSCHEMA	VARCHAR (128)		Schema name of the table to which the constraint applies.
TABNAME	VARCHAR (128)		Unqualified name of the table to which the constraint applies.
BTYPE	CHAR (1)		Type of object on which the constraint depends. Possible values are: • F = Routine • I = Index • R = User-defined structured type • u = Module alias
BSCHEMA	VARCHAR (128)		Schema name of the object on which the constraint depends.
BMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the object on which a dependency belongs. The null value if not a module object.
BNAME	VARCHAR (128)		Unqualified name of the object on which the constraint depends.

Table 190. SYSCAT.CONSTDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
BMODULEID	INTEGER	Y	Identifier for the module of the object on which the constraint depends.

SYSCAT.CONTEXTATTRIBUTES

Each row represents a trusted context attribute.

Table 191. SYSCAT.CONTEXTATTRIBUTES Catalog View

Column Name	Data Type	Nullable	Description
CONTEXTNAME	VARCHAR (128)		Name of the trusted context.
ATTR_NAME	VARCHAR (128)		Name of the attribute. One of: • ADDRESS • ENCRYPTION
ATTR_VALUE	VARCHAR (128)		Value of the attribute.
ATTR_OPTIONS	VARCHAR (128)	Y	If ATTR_NAME is 'ADDRESS', specifies the level of encryption required for this specific address. A null value indicates that the global ENCRYPTION attribute applies.

SYSCAT.CONTEXTS

Each row represents a trusted context.

Table 192. SYSCAT.CONTEXTS Catalog View

Column Name	Data Type	Nullable	Description
CONTEXTNAME	VARCHAR (128)		Name of the trusted context.
CONTEXTID	INTEGER		Identifier for the trusted context.
SYSTEMAUTHID	VARCHAR (128)		The system authorization ID associated with the trusted context.
DEFAULTCONTEXTROLE	VARCHAR (128)	Y	The default role for the context.
CREATE_TIME	TIMESTAMP		Time at which the trusted context was created.
ALTER_TIME	TIMESTAMP		Time at which the trusted context was last altered.
ENABLED	CHAR (1)		Trusted context state. • N = Disabled • Y = Enabled
AUDITPOLICYID	INTEGER	Y	Identifier for the audit policy.
AUDITPOLICYNAME	VARCHAR (128)	Y	Name of the audit policy.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.DATAPARTITIONEXPRESSION

Each row represents an expression for that part of the table partitioning key.

Table 193. SYSCAT.DATAPARTITIONEXPRESSION Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the partitioned table.
TABNAME	VARCHAR (128)		Unqualified name of the partitioned table.
DATAPARTITIONKEYSEQ	INTEGER		Expression key part sequence ID, starting from 1.
DATAPARTITIONEXPRESSION	CLOB (32K)		Expression for this entry in the sequence, in SQL syntax.
NULLSFIRST	CHAR (1)		• N = Null values in this expression compare high • Y = Null values in this expression compare low

SYSCAT.DATAPARTITIONS

Each row represents a data partition. Note:

- The data partition statistics represent one database partition if the table is created on multiple database partitions.

Table 194. SYSCAT.DATAPARTITIONS Catalog View

Column Name	Data Type	Nullable	Description
DATAPARTITIONNAME	VARCHAR (128)		Name of the data partition.
TABSCHEMA	VARCHAR (128)		Schema name of the table to which this data partition belongs.
TABNAME	VARCHAR (128)		Unqualified name of the table to which this data partition belongs.
DATAPARTITIONID	INTEGER		Identifier for the data partition.
TBSPACEID	INTEGER	Y	Identifier for the table space in which this data partition is stored. The null value when STATUS is 'I'.
PARTITIONOBJECTID	INTEGER	Y	Identifier for the data partition within the table space.
LONG_TBSPACEID	INTEGER	Y	Identifier for the table space in which long data is stored. The null value when STATUS is 'I'.
ACCESS_MODE	CHAR (1)		Access restriction state of the data partition. These states only apply to objects that are in set integrity pending state or to objects that were processed by a SET INTEGRITY statement. Possible values are: • D = No data movement • F = Full access • N = No access • R = Read-only access

Table 194. SYSCAT.DATAPARTITIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
STATUS	VARCHAR (32)		• A = Data partition is newly attached • D = Data partition is detached • I = Detached data partition whose entry in the catalog is maintained only during asynchronous index cleanup; rows with a STATUS value of 'I' are removed when all index records referring to the detached partition have been deleted • Empty string = Data partition is visible (normal status) Bytes 2 through 32 are reserved for future use.
SEQNO	INTEGER		Data partition sequence number (starting from 0).
LOWINCLUSIVE	CHAR (1)		• N = Low key value is not inclusive • Y = Low key value is inclusive
LOWVALUE	VARCHAR (512)		Low key value (a string representation of an SQL value) for this data partition.
HIGHINCLUSIVE	CHAR (1)		• N = High key value is not inclusive • Y = High key value is inclusive
HIGHVALUE	VARCHAR (512)		High key value (a string representation of an SQL value) for this data partition.
CARD	BIGINT		Total number of rows in the data partition; -1 if statistics are not collected.
OVERFLOW	BIGINT		Total number of overflow records in the data partition; -1 if statistics are not collected.
NPAGES	BIGINT		Total number of pages on which the rows of the data partition exist; -1 if statistics are not collected.
FPAGES	BIGINT		Total number of pages in the data partition; -1 if statistics are not collected.
ACTIVE_BLOCKS	BIGINT		Total number of active blocks in the data partition, or -1. Applies to multidimensional clustering (MDC) tables only.
INDEX_TBSPACEID	INTEGER		Identifier for the table space which holds all partitioned indexes for this data partition.
AVGROWSIZE	SMALLINT		Average length (in bytes) of both compressed and uncompressed rows in this data partition; -1 if statistics are not collected.
PCTROWSCOMPRESSED	REAL		Compressed rows as a percentage of the total number of rows in the data partition; -1 if statistics are not collected.
PCTPAGESAVED	SMALLINT		Approximate percentage of pages saved in the data partition as a result of row compression. This value includes overhead bytes for each user data row in the data partition, but does not include the space that is consumed by dictionary overhead; -1 if statistics are not collected.

Table 194. SYSCAT.DATAPARTITIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
AVGCOMPRESSEDROWSIZE	SMALLINT		Average length (in bytes) of compressed rows in this data partition; -1 if statistics are not collected.
AVGROWCOMPRESSIONRATIO	REAL		For compressed rows in the data partition, this is the average compression ratio by row; that is, the average uncompressed row length divided by the average compressed row length; -1 if statistics are not collected.
STATS_TIME	TIMESTAMP	Y	Time at which any change was last made to recorded statistics for this object. Null if statistics are not collected.
LASTUSED	DATE		Reserved for future use.

SYSCAT.DATATYPEDEP

Each row represents a dependency of a user-defined data type on some other object.

Table 195. SYSCAT.DATATYPEDEP Catalog View

Column Name	Data Type	Nullable	Description
TYPESCHEMA	VARCHAR (128)		Schema name of the data type.
TYPEMODULENAME	VARCHAR (128)	Y	Module name of the data type.
TYPENAME	VARCHAR (128)		Unqualified name of the data type.
TYPEMODULEID	INTEGER	Y	Identifier for the module of the data type.
BTYPE	CHAR (1)		Type of object on which there is a dependency. Possible values are: • A = Table alias • G = Global temporary table • H = Hierarchy table • N = Nickname • R = User-defined data type • S = Materialized query table • T = Table (not typed) • U = Typed table • V = View (not typed) • W = Typed view • q = Sequence alias • u = Module alias • v = Global variable • * = Anchored to the row of a base table
BSCHEMA	VARCHAR (128)		Schema name of the object on which there is a dependency.
BMODULENAME	VARCHAR (128)	Y	Module name of the object on which there is a dependency.
BNAME	VARCHAR (128)		Unqualified name of the object on which there is a dependency.

Table 195. SYSCAT.DATATYPEDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
BMODULEID	INTEGER	Y	Identifier for the module of the object on which there is a dependency.
TABAUTH	SMALLINT	Y	If BTYPE = 'S', 'T', 'U', 'V', 'W', or 'v', encodes the privileges on the table or view that are required by the dependent data type; the null value otherwise.

SYSCAT.DATATYPES

Each row represents a built-in or user-defined data type.

Table 196. SYSCAT.DATATYPES Catalog View

Column Name	Data Type	Nullable	Description
TYPESCHEMA	VARCHAR (128)		Schema name of the data type if TYPEMODULEID is null; otherwise schema name of the module to which the data type belongs.
TYPEMODULENAME	VARCHAR (128)	Y	Unqualified name of the module to which the user-defined type belongs. The null value if not a module user-defined type.
TYPENAME	VARCHAR (128)		Unqualified name of the data type.
OWNER	VARCHAR (128)		Authorization ID of the owner of the type.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
SOURCESCHEMA	VARCHAR (128)	Y	For distinct types or array types, the schema name of the source data type. For user-defined structured types, the schema name of the built-in type of the reference representation type. Null for other data types.
SOURCEMODULENAME	VARCHAR (128)	Y	Unqualified name of the module to which the source data type belongs. The null value if not a module source data type.
SOURCENAME	VARCHAR (128)	Y	For distinct types or array types, the unqualified name of the source data type. For user-defined structured types, the unqualified built-in type name of the reference representation type. Null for other data types.
METATYPE	CHAR (1)		• A = User-defined array type • C = User-defined cursor type • F = User-defined row type • L = User-defined associative array type • R = User-defined structured type • S = System predefined type • T = User-defined distinct type
TYPEID	SMALLINT		Identifier for the data type.

Table 196. SYSCAT.DATATYPES Catalog View (continued)

Column Name	Data Type	Nullable	Description
TYPEMODULEID	INTEGER	Y	Identifier for the module to which the user-defined type belongs. The null value if not a module user-defined type.
SOURCETYPEID	SMALLINT	Y	Identifier for the source type (the null value for built-in types). For user-defined structured types, this is the identifier of the reference representation type.
SOURCEMODULEID	INTEGER	Y	Identifier for the module to which the source data type belongs. The null value if not a module source data type.
PUBLISHED	CHAR (1)		Indicates whether the module user-defined type can be referenced outside its module. • N = The module user-defined type is not published • Y = The module user-defined type is published • Blank = Not applicable
LENGTH	INTEGER		Maximum length of the type. 0 for built-in parameterized types (for example, DECIMAL and VARCHAR). For user-defined structured types, this is the length of the reference representation type.
SCALE	SMALLINT		Scale for distinct types or reference representation types based on the built-in DECIMAL type; the number of digits of fractional seconds for distinct types based on the built-in TIMESTAMP type; 6 for the built-in TIMESTAMP type; 0 for all other types (including DECIMAL itself).
CODEPAGE	SMALLINT		Database code page for string types, distinct types based on string types, or reference representation types; 0 otherwise.
COLLATIONSCHEMA	VARCHAR (128)	Y	For string types, the schema name of the collation for the data type; the null value otherwise.
COLLATIONNAME	VARCHAR (128)	Y	For string types, the unqualified name of the collation for the data type; the null value otherwise.
ARRAY_LENGTH	INTEGER	Y	Maximum cardinality of the array. The null value if METATYPE is not 'A'.
ARRAYINDEXTYPESCHEMA	VARCHAR(128)	Y	Schema of the data type of the array index. The null value if METATYPE is not 'L'.
ARRAYINDEXTYPENAME	VARCHAR (128)	Y	Name of the data type of the array index. The null value if METATYPE is not 'L'.
ARRAYINDEXTYPEID	SMALLINT	Y	Identifier for the array index type. The null value if METATYPE is not 'L'.
ARRAYINDEXTYPELENGTH	INTEGER	Y	Maximum length of the array index data type. The null value if METATYPE is not 'L'.
CREATE_TIME	TIMESTAMP		Creation time of the data type.

Table 196. SYSCAT.DATATYPES Catalog View (continued)

Column Name	Data Type	Nullable	Description
VALID	CHAR (1)		• N = The data type is invalid • Y = The data type is valid
ATTRCOUNT	SMALLINT		Number of attributes in the data type.
INSTANTIABLE	CHAR (1)		• N = Type cannot be instantiated • Y = Type can be instantiated
WITH_FUNC_ACCESS	CHAR (1)		• N = Methods for this type cannot be invoked using function notation. • Y = All the methods for this type can be invoked using function notation.
FINAL	CHAR (1)		• N = The user-defined type can have subtypes. • Y = The user-defined type cannot have subtypes.
INLINE_LENGTH	INTEGER		Maximum length of a structured type that can be kept with a base table row; 0 otherwise.
NATURAL_ INLINE_LENGTH	INTEGER	Y	System-generated natural inline length of a structured type instance. The null value if this type is not a structured type.
JARSCHEMA	VARCHAR (128)	Y	Schema name of the JAR_ID that identifies the Jar file containing the Java class that implements the SQL type. The null value if the EXTERNAL NAME clause is not specified.
JAR_ID	VARCHAR (128)	Y	Identifier for the Jar file that contains the Java class that implements the SQL type. The null value if the EXTERNAL NAME clause is not specified.
CLASS	VARCHAR (384)	Y	Java class that implements the SQL type. The null value if the EXTERNAL NAME clause is not specified.
SQLJ_REPRESENTATION	CHAR (1)	Y	SQLJ ″representation_spec″ of the Java class that implements the SQL type. The null value if the EXTERNAL NAME ... LANGUAGE JAVA REPRESENTATION SPEC clause is not specified. • D = SQL data • S = Serializable
ALTER_TIME	TIMESTAMP		Time at which the data type was last altered.
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the type.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.DBPARTITIONGROUPDEF

Each row represents a database partition that is contained in a database partition group.

Table 197. SYSCAT.DBPARTITIONGROUPDEF Catalog View

Column Name	Data Type	Nullable	Description
DBPGNAME	VARCHAR (128)		Name of the database partition group that contains the database partition.
DBPARTITIONNUM	SMALLINT		Partition number of a database partition that is contained in the database partition group. A valid partition number is between 0 and 999, inclusive.
IN_USE	CHAR (1)		Status of the database partition. • A = The newly added database partition is not in the distribution map, but the containers for the table spaces in the database partition group have been created; the database partition is added to the distribution map when a redistribute database partition group operation has completed successfully. • D = The database partition will be dropped when a redistribute database partition group operation has completed successfully. • T = The newly added database partition is not in the distribution map, and it was added using the WITHOUT TABLESPACES clause; containers must be added to the table spaces in the database partition group. • Y = The database partition is in the distribution map.

SYSCAT.DBPARTITIONGROUPS

Each row represents a database partition group.

Table 198. SYSCAT.DBPARTITIONGROUPS Catalog View

Column Name	Data Type	Nullable	Description
DBPGNAME	VARCHAR (128)		Name of the database partition group.
OWNER	VARCHAR (128)		Authorization ID of the owner of the database partition group.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
PMAP_ID	SMALLINT		Identifier for the distribution map in the SYSCAT.PARTITIONMAPS catalog view.
REDISTRIBUTE_PMAP_ID	SMALLINT		Identifier for the distribution map currently being used for redistribution; -1 if redistribution is currently not in progress.

Table 198. SYSCAT.DBPARTITIONGROUPS Catalog View (continued)

Column Name	Data Type	Nullable	Description
CREATE_TIME	TIMESTAMP		Creation time of the database partition group.
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the database partition group.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.EVENTMONITORS

Each row represents an event monitor.

Table 199. SYSCAT.EVENTMONITORS Catalog View

Column Name	Data Type	Nullable	Description
EVMONNAME	VARCHAR (128)		Name of the event monitor.
OWNER	VARCHAR (128)		Authorization ID of the owner of the event monitor.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
TARGET_TYPE	CHAR (1)		Type of target to which event data is written. • F = File • P = Pipe • T = Table • U = Unformatted event table
TARGET	VARCHAR (762)		Name of the target to which file or pipe event monitor data is written. For files, it can be either an absolute path name or a relative path name (relative to the database path for the database; this can be seen using the LIST ACTIVE DATABASES command). For pipes, it can be an absolute path name.
MAXFILES	INTEGER	Y	Maximum number of event files that this event monitor permits in an event path. The null value if there is no maximum, or if TARGET_TYPE is not 'F' (file).
MAXFILESIZE	INTEGER	Y	Maximum size (in 4K pages) that each event file can attain before the event monitor creates a new file. The null value if there is no maximum, or if TARGET_TYPE is not 'F' (file).
BUFFERSIZE	INTEGER	Y	Size of the buffer (in 4K pages) that is used by event monitors with file targets; null value otherwise.
IO_MODE	CHAR (1)	Y	Mode of file input/output (I/O). • B = Blocked • N = Not blocked • Null value = TARGET_TYPE is not 'F' (file) or 'T' (table)

Table 199. SYSCAT.EVENTMONITORS Catalog View (continued)

Column Name	Data Type	Nullable	Description
WRITE_MODE	CHAR (1)	Y	Indicates how this event monitor handles existing event data when the monitor is activated. • A = Append • R = Replace • Null value = TARGET_TYPE is not 'F' (file)
AUTOSTART	CHAR (1)		Indicates whether this event monitor is to be activated automatically when the database starts. • N = No • Y = Yes
DBPARTITIONNUM	SMALLINT		Number of the database partition where the event monitor runs and logs events.
MONSCOPE	CHAR (1)		Monitoring scope. • G = Global • L = Local • T = Each database partition on which the table space exists • Blank = WRITE TO TABLE event monitor
EVMON_ACTIVATES	INTEGER		Number of times the event monitor has been activated.
NODENUM[1]	SMALLINT		Number of the database partition where the event monitor runs and logs events.
DEFINER[2]	VARCHAR (128)		Authorization ID of the owner of the event monitor.
REMARKS	VARCHAR (254)	Y	Reserved for future use.

Note:

1. The NODENUM column is included for backwards compatibility. See DBPARTITIONNUM.
2. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.EVENTS

Each row represents an event that is being monitored. An event monitor, in general, monitors multiple events.

Table 200. SYSCAT.EVENTS Catalog View

Column Name	Data Type	Nullable	Description
EVMONNAME	VARCHAR (128)		Name of the event monitor that is monitoring this event.

Table 200. SYSCAT.EVENTS Catalog View (continued)

Column Name	Data Type	Nullable	Description
TYPE	VARCHAR (128)		Type of event being monitored. Possible values are: • ACTIVITIES • CONNECTIONS • DATABASE • DEADLOCKS • DETAILDEADLOCKS • LOCKING • PACKAGECACHESTMT • STATEMENTS • TABLES • TABLESPACES • THRESHOLD_VIOLATIONS • TRANSACTIONS • STATISTICS • UOW
FILTER	CLOB (64K)	Y	Full text of the WHERE clause that applies to this event.

SYSCAT.EVENTTABLES

Each row represents the target table of an event monitor that writes to SQL tables.

Table 201. SYSCAT.EVENTTABLES Catalog View

Column Name	Data Type	Nullable	Description
EVMONNAME	VARCHAR (128)		Name of the event monitor.

Table 201. SYSCAT.EVENTTABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
LOGICAL_GROUP	VARCHAR (128)		Name of the logical data group. Possible values are: • ACTIVITYHISTORY • BUFFERPOOL • CONN • CONNHEADER • CONTROL • DATAVAL • DB • DEADLOCK • DLCONN • DLLOCK • LOCKING • PACKAGECACHESTMT • SCSTATS • STMT • STMTHIST • STMTVALS • SUBSECTION • TABLE • TABLESPACE • THRESHOLDVIOLATIONS • UOW • WCSTATS • WLSTATS • XACT
TABSCHEMA	VARCHAR (128)		Schema name of the target table.
TABNAME	VARCHAR (128)		Unqualified name of the target table.
PCTDEACTIVATE	SMALLINT		A percent value that specifies how full a DMS table space must be before an event monitor automatically deactivates. Set to 100 for SMS table spaces.

SYSCAT.FULLHIERARCHIES

Each row represents the relationship between a subtable and a supertable, a subtype and a supertype, or a subview and a superview. All hierarchical relationships, including immediate ones, are included in this view.

Table 202. SYSCAT.FULLHIERARCHIES Catalog View

Column Name	Data Type	Nullable	Description
METATYPE	CHAR (1)		Relationship type. • R = Between structured types • U = Between typed tables • W = Between typed views

Table 202. SYSCAT.FULLHIERARCHIES Catalog View (continued)

Column Name	Data Type	Nullable	Description
SUB_SCHEMA	VARCHAR (128)		Schema name of the subtype, subtable, or subview.
SUB_NAME	VARCHAR (128)		Unqualified name of the subtype, subtable, or subview.
SUPER_SCHEMA	VARCHAR (128)	Y	Schema name of the supertype, supertable, or superview.
SUPER_NAME	VARCHAR (128)	Y	Unqualified name of the supertype, supertable, or superview.
ROOT_SCHEMA	VARCHAR (128)		Schema name of the table, view, or type that is at the root of the hierarchy.
ROOT_NAME	VARCHAR (128)		Unqualified name of the table, view, or type that is at the root of the hierarchy.

SYSCAT.FUNCMAPOPTIONS

Each row represents a function mapping option value.

Table 203. SYSCAT.FUNCMAPOPTIONS Catalog View

Column Name	Data Type	Nullable	Description
FUNCTION_MAPPING	VARCHAR (128)		Name of the function mapping.
OPTION	VARCHAR (128)		Name of the function mapping option.
SETTING	VARCHAR (2048)		Value of the function mapping option.

SYSCAT.FUNCMAPPARMOPTIONS

Each row represents a function mapping parameter option value.

Table 204. SYSCAT.FUNCMAPPARMOPTIONS Catalog View

Column Name	Data Type	Nullable	Description
FUNCTION_MAPPING	VARCHAR (128)		Name of the function mapping.
ORDINAL	SMALLINT		Position of the parameter.
LOCATION	CHAR (1)		Location of the parameter. • L = Local parameter • R = Remote parameter
OPTION	VARCHAR (128)		Name of the function mapping parameter option.
SETTING	VARCHAR (2048)		Value of the function mapping parameter option.

SYSCAT.FUNCMAPPINGS

Each row represents a function mapping.

Table 205. SYSCAT.FUNCMAPPINGS Catalog View

Column Name	Data Type	Nullable	Description
FUNCTION_MAPPING	VARCHAR (128)		Name of the function mapping (might be system-generated).
FUNCSCHEMA	VARCHAR (128)	Y	Schema name of the function. If the null value, the function is assumed to be a built-in function.
FUNCNAME	VARCHAR (1024)	Y	Unqualified name of the user-defined or built-in function.
FUNCID	INTEGER	Y	Identifier for the function.
SPECIFICNAME	VARCHAR (128)	Y	Name of the routine instance (might be system-generated).
OWNER	VARCHAR (128)		Authorization ID of the owner of the mapping. 'SYSIBM' indicates that this is a built-in function.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
WRAPNAME	VARCHAR (128)	Y	Wrapper to which this mapping applies.
SERVERNAME	VARCHAR (128)	Y	Name of the data source.
SERVERTYPE	VARCHAR (30)	Y	Type of data source to which this mapping applies.
SERVERVERSION	VARCHAR (18)	Y	Version of the server type to which this mapping applies.
CREATE_TIME	TIMESTAMP		Time at which the mapping was created.
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the mapping. 'SYSIBM' indicates that this is a built-in function.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.HIERARCHIES

Each row represents the relationship between a subtable and its immediate supertable, a subtype and its immediate supertype, or a subview and its immediate superview. Only immediate hierarchical relationships are included in this view.

Table 206. SYSCAT.HIERARCHIES Catalog View

Column Name	Data Type	Nullable	Description
METATYPE	CHAR (1)		Relationship type. • R = Between structured types • U = Between typed tables • W = Between typed views
SUB_SCHEMA	VARCHAR (128)		Schema name of the subtype, subtable, or subview.
SUB_NAME	VARCHAR (128)		Unqualified name of the subtype, subtable, or subview.

Table 206. SYSCAT.HIERARCHIES Catalog View (continued)

Column Name	Data Type	Nullable	Description
SUPER_SCHEMA	VARCHAR (128)		Schema name of the supertype, supertable, or superview.
SUPER_NAME	VARCHAR (128)		Unqualified name of the supertype, supertable, or superview.
ROOT_SCHEMA	VARCHAR (128)		Schema name of the table, view, or type that is at the root of the hierarchy.
ROOT_NAME	VARCHAR (128)		Unqualified name of the table, view, or type that is at the root of the hierarchy.

SYSCAT.HISTOGRAMTEMPLATEBINS

Each row represents a histogram template bin.

Table 207. SYSCAT.HISTOGRAMTEMPLATEBINS Catalog View

Column Name	Data Type	Nullable	Description
TEMPLATENAME	VARCHAR (128)	Y	Name of the histogram template.
TEMPLATEID	INTEGER		Identifier for the histogram template.
BINID	INTEGER		Identifier for the histogram template bin.
BINUPPERVALUE	BIGINT		The upper value for a single bin in the histogram template.

SYSCAT.HISTOGRAMTEMPLATES

Each row represents a histogram template.

Table 208. SYSCAT.HISTOGRAMTEMPLATES Catalog View

Column Name	Data Type	Nullable	Description
TEMPLATEID	INTEGER		Identifier for the histogram template.
TEMPLATENAME	VARCHAR (128)		Name of the histogram template.
CREATE_TIME	TIMESTAMP		Time at which the histogram template was created.
ALTER_TIME	TIMESTAMP		Time at which the histogram template was last altered.
NUMBINS	INTEGER		Number of bins in the histogram template, including the last bin that has an unbounded top value.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.HISTOGRAMTEMPLATEUSE

Each row represents a relationship between a workload management object that can use histogram templates and a histogram template.

Table 209. SYSCAT.HISTOGRAMTEMPLATEUSE Catalog View

Column Name	Data Type	Nullable	Description
TEMPLATENAME	VARCHAR (128)	Y	Name of the histogram template.

Table 209. SYSCAT.HISTOGRAMTEMPLATEUSE Catalog View (continued)

Column Name	Data Type	Nullable	Description
TEMPLATEID	INTEGER		Identifier for the histogram template.
HISTOGRAMTYPE	CHAR (1)		The type of information collected by histograms based on this template. • C = Activity estimated cost histogram • E = Activity execution time histogram • I = Activity interarrival time histogram • L = Activity life time histogram • Q = Activity queue time histogram • R = Request execution time histogram
OBJECTTYPE	CHAR (1)		The type of WLM object. • b = Service class • k = Work action • w = Workload
OBJECTID	INTEGER		Identifier of the WLM object.
SERVICECLASSNAME	VARCHAR (128)	Y	Name of the service class.
PARENTSERVICECLASSNAME	VARCHAR (128)	Y	The name of the parent service class of the service subclass that uses the histogram template.
WORKACTIONNAME	VARCHAR (128)	Y	The name of the work action that uses the histogram template.
WORKACTIONSETNAME	VARCHAR (128)	Y	The name of the work action set containing the work action that uses the histogram template.
WORKLOADNAME	VARCHAR (128)	Y	The name of the workload that uses the histogram template.

SYSCAT.INDEXCOLUSE

Each row represents a column that participates in an index.

Table 210. SYSCAT.INDEXCOLUSE Catalog View

Column Name	Data Type	Nullable	Description
INDSCHEMA	VARCHAR (128)		Schema name of the index.
INDNAME	VARCHAR (128)		Unqualified name of the index.
COLNAME	VARCHAR (128)		Name of the column.
COLSEQ	SMALLINT		Numeric position of the column in the index (initial position is 1).
COLORDER	CHAR (1)		Order of the values in this index column. Possible values are: • A = Ascending • D = Descending • I = INCLUDE column (ordering ignored)
COLLATIONSCHEMA	VARCHAR (128)	Y	For string types, the schema name of the collation for the column; the null value otherwise.

Table 210. SYSCAT.INDEXCOLUSE Catalog View (continued)

Column Name	Data Type	Nullable	Description
COLLATIONNAME	VARCHAR (128)	Y	For string types, the unqualified name of the collation for the column; the null value otherwise.

SYSCAT.INDEXDEP

Each row represents a dependency of an index on some other object. The index depends on an object of type BTYPE and name BNAME, so a change to the object affects the index.

Table 211. SYSCAT.INDEXDEP Catalog View

Column Name	Data Type	Nullable	Description
INDSCHEMA	VARCHAR (128)		Schema name of the index.
INDNAME	VARCHAR (128)		Unqualified name of the index.
BTYPE	CHAR (1)		Type of object on which there is a dependency. Possible values are: • A = Table alias • B = Trigger • F = Routine • G = Global temporary table • H = Hierachy table • K = Package • L = Detached table • N = Nickname • O = Privilege dependency on all subtables or subviews in a table or view hierarchy • Q = Sequence • R = User-defined data type • S = Materialized query table • T = Table (not typed) • U = Typed table • V = View (not typed) • W = Typed view • X = Index extension • Z = XSR object • q = Sequence alias • u = Module alias • v = Global variable • * = Anchored to the row of a base table
BSCHEMA	VARCHAR (128)		Schema name of the object on which there is a dependency.
BMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the object on which a dependency belongs. The null value if not a module object.
BNAME	VARCHAR (128)		Unqualified name of the object on which there is a dependency. For routines (BTYPE = 'F'), this is the specific name.

Table 211. SYSCAT.INDEXDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
BMODULEID	INTEGER	Y	Identifier for the module of the object on which there is a dependency.
TABAUTH	SMALLINT	Y	If BTYPE = 'O', 'S', 'T', 'U', 'V', 'W', or 'v', encodes the privileges on the table or view that are required by the dependent index; the null value otherwise.

SYSCAT.INDEXES

Each row represents an index. Indexes on typed tables are represented by two rows: one for the "logical index" on the typed table, and one for the "H-index" on the hierarchy table.

Table 212. SYSCAT.INDEXES Catalog View

Column Name	Data Type	Nullable	Description
INDSCHEMA	VARCHAR (128)		Schema name of the index.
INDNAME	VARCHAR (128)		Unqualified name of the index.
OWNER	VARCHAR (128)		Authorization ID of the owner of the index.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
TABSCHEMA	VARCHAR (128)		Schema name of the table or nickname on which the index is defined.
TABNAME	VARCHAR (128)		Unqualified name of the table or nickname on which the index is defined.
COLNAMES	VARCHAR (640)		This column is no longer used and will be removed in the next release. Use SYSCAT.INDEXCOLUSE for this information.
UNIQUERULE	CHAR (1)		Unique rule. • D = Permits duplicates • U = Unique • P = Implements primary key
MADE_UNIQUE	CHAR (1)		• N = Index remains as it was created • Y = This index was originally non-unique but was converted to a unique index to support a unique or primary key constraint. If the constraint is dropped, the index reverts to being non-unique.
COLCOUNT	SMALLINT		Number of columns in the key, plus the number of include columns, if any.
UNIQUE_COLCOUNT	SMALLINT		Number of columns required for a unique key. It is always <= COLCOUNT, and < COLCOUNT only if there are include columns; -1 if the index has no unique key (that is, it permits duplicates).

Table 212. SYSCAT.INDEXES Catalog View (continued)

Column Name	Data Type	Nullable	Description
INDEXTYPE[5]	CHAR (4)		Type of index. • BLOK = Block index • CLUS = Clustering index (controls the physical placement of newly inserted rows) • DIM = Dimension block index • REG = Regular index • XPTH = XML path index • XRGN = XML region index • XVIL = Index over XML column (logical) • XVIP = Index over XML column (physical)
ENTRYTYPE	CHAR (1)		• H = This row represents an index on a hierarchy table • L = This row represents a logical index on a typed table • Blank = This row represents an index on an untyped table
PCTFREE	SMALLINT		Percentage of each index page to be reserved during the initial building of the index. This space is available for data insertions after the index has been built.
IID	SMALLINT		Identifier for the index.
NLEAF	BIGINT		Number of leaf pages; -1 if statistics are not collected.
NLEVELS	SMALLINT		Number of index levels; -1 if statistics are not collected.
FIRSTKEYCARD	BIGINT		Number of distinct first-key values; -1 if statistics are not collected.
FIRST2KEYCARD	BIGINT		Number of distinct keys using the first two columns of the index; -1 if statistics are not collected, or if not applicable.
FIRST3KEYCARD	BIGINT		Number of distinct keys using the first three columns of the index; -1 if statistics are not collected, or if not applicable.
FIRST4KEYCARD	BIGINT		Number of distinct keys using the first four columns of the index; -1 if statistics are not collected, or if not applicable.
FULLKEYCARD	BIGINT		Number of distinct full-key values; -1 if statistics are not collected.
CLUSTERRATIO[3]	SMALLINT		Degree of data clustering with the index; -1 if statistics are not collected or if detailed index statistics are collected (in which case, CLUSTERFACTOR will be used instead).
CLUSTERFACTOR[3]	DOUBLE		Finer measurement of the degree of clustering; -1 if statistics are not collected or if the index is defined on a nickname.

Table 212. SYSCAT.INDEXES Catalog View (continued)

Column Name	Data Type	Nullable	Description
SEQUENTIAL_PAGES	BIGINT		Number of leaf pages located on disk in index key order with few or no large gaps between them; -1 if statistics are not collected.
DENSITY	INTEGER		Ratio of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index, expressed as a percent (integer between 0 and 100); -1 if statistics are not collected.
USER_DEFINED	SMALLINT		1 if this index was defined by a user and has not been dropped; 0 otherwise.
SYSTEM_REQUIRED	SMALLINT		• 1 if one or the other of the following conditions is met: – This index is required for a primary or unique key constraint, or this index is a dimension block index or composite block index for a multidimensional clustering (MDC) table. - This is the index on the object identifier (OID) column of a typed table. • 2 if both of the following conditions are met: – This index is required for a primary or unique key constraint, or this index is a dimension block index or composite block index for an MDC table. - This is the index on the OID column of a typed table. • 0 otherwise.
CREATE_TIME	TIMESTAMP		Time when the index was created.
STATS_TIME	TIMESTAMP	Y	Last time that any change was made to the recorded statistics for this index. The null value if no statistics are available.
PAGE_FETCH_PAIRS[3]	VARCHAR (520)		A list of pairs of integers, represented in character form. Each pair represents the number of pages in a hypothetical buffer, and the number of page fetches required to scan the table with this index using that hypothetical buffer. Zero-length string if no data is available.
MINPCTUSED	SMALLINT		A non-zero integer value indicates that the index is enabled for online defragmentation, and represents the minimum percentage of used space on a page before a page merge can be attempted. A zero value indicates that no page merge is attempted.
REVERSE_SCANS	CHAR (1)		• N = Index does not support reverse scans • Y = Index supports reverse scans

Table 212. SYSCAT.INDEXES Catalog View (continued)

Column Name	Data Type	Nullable	Description
INTERNAL_FORMAT	SMALLINT		Possible values are: • 1 = Index does not have backward pointers • 2 or greater = Index has backward pointers • 6 = Index is a composite block index
COMPRESSION	CHAR (1)		Specifies whether index compression is activated • N = Not activated • Y = Activated
IESCHEMA	VARCHAR (128)	Y	Schema name of the index extension. The null value for ordinary indexes.
IENAME	VARCHAR (128)	Y	Unqualified name of the index extension. The null value for ordinary indexes.
IEARGUMENTS	CLOB (64K)	Y	External information of the parameter specified when the index is created. The null value for ordinary indexes.
INDEX_OBJECTID	INTEGER		Identifier for the index object.
NUMRIDS	BIGINT		Total number of row identifiers (RIDs) or block identifiers (BIDs) in the index; -1 if not known.
NUMRIDS_DELETED	BIGINT		Total number of row identifiers (or block identifiers) in the index that are marked deleted, excluding those identifiers on leaf pages on which all the identifiers are marked deleted.
NUM_EMPTY_LEAFS	BIGINT		Total number of index leaf pages that have all of their row identifiers (or block identifiers) marked deleted.
AVERAGE_RANDOM_FETCH_PAGES[1,2]	DOUBLE		Average number of random table pages between sequential page accesses when fetching using the index; -1 if not known.
AVERAGE_RANDOM_PAGES[2]	DOUBLE		Average number of random table pages between sequential page accesses; -1 if not known.
AVERAGE_SEQUENCE_GAP[2]	DOUBLE		Gap between index page sequences. Detected through a scan of index leaf pages, each gap represents the average number of index pages that must be randomly fetched between sequences of index pages; -1 if not known.
AVERAGE_SEQUENCE_FETCH_GAP[1,2]	DOUBLE		Gap between table page sequences when fetching using the index. Detected through a scan of index leaf pages, each gap represents the average number of table pages that must be randomly fetched between sequences of table pages; -1 if not known.

Table 212. SYSCAT.INDEXES Catalog View (continued)

Column Name	Data Type	Nullable	Description
AVERAGE_SEQUENCE_PAGES[2]	DOUBLE		Average number of index pages that are accessible in sequence (that is, the number of index pages that the prefetchers would detect as being in sequence); -1 if not known.
AVERAGE_SEQUENCE_FETCH_ PAGES[1,2]	DOUBLE		Average number of table pages that are accessible in sequence (that is, the number of table pages that the prefetchers would detect as being in sequence) when fetching using the index; -1 if not known.
TBSPACEID	INTEGER		Identifier for the index table space.
LEVEL2PCTFREE	SMALLINT		Percentage of each index level 2 page to be reserved during initial building of the index. This space is available for future inserts after the index has been built.
PAGESPLIT	CHAR (1)		Index page split behavior. • H = High • L = Low • S = Symmetric
AVGPARTITION_ CLUSTERRATIO[3]	SMALLINT		Degree of data clustering within a single data partition. -1 if the table is not partitioned, if statistics are not collected, or if detailed statistics are collected (in which case AVGPARTITION_ CLUSTERFACTOR will be used instead).
AVGPARTITION_ CLUSTERFACTOR[3]	DOUBLE		Finer measurement of the degree of clustering within a single data partition. -1 if the table is not partitioned, if statistics are not collected, or if the index is defined on a nickname.
AVGPARTITION_PAGE_FETCH_ PAIRS[3]	VARCHAR (520)		A list of paired integers in character form. Each pair represents a potential buffer pool size and the corresponding page fetches required to access a single data partition from the table. Zero-length string if no data is available, or if the table is not partitioned.
PCTPAGESSAVED	SMALLINT		Approximate percentage of pages saved in the index as a result of index compression. -1 if statistics are not collected.
DATAPARTITION_ CLUSTERFACTOR	DOUBLE		A statistic measuring the ″clustering″ of the index keys with regard to data partitions. It is a number between 0 and 1, with 1 representing perfect clustering and 0 representing no clustering.
INDCARD	BIGINT		Cardinality of the index. This might be different from the cardinality of the table for indexes that do not have a one-to-one relationship between the table rows and the index entries.
AVGLEAFKEYSIZE	INTEGER		Average index key size for keys on leaf pages in the index.

Table 212. SYSCAT.INDEXES Catalog View (continued)

Column Name	Data Type	Nullable	Description
AVGNLEAFKEYSIZE	INTEGER		Average index key size for keys on non-leaf pages in the index.
OS_PTR_SIZE	INTEGER		Platform word size with which the index was created. • 32 = 32-bit • 64 = 64-bit
COLLECTSTATISTICS	CHAR (1)		Specifies how statistics were collected at index creation time. • D = Collect detailed index statistics • S = Collect sampled detailed index statistics • Y = Collect basic index statistics • Blank = Do not collect index statistics
DEFINER[4]	VARCHAR (128)		Authorization ID of the owner of the index.
LASTUSED	DATE		Reserved for future use.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. When using DMS table spaces, this statistic cannot be computed.
2. Prefetch statistics are not gathered during a LOAD...STATISTICS YES, or a CREATE INDEX...COLLECT STATISTICS operation, or when the database configuration parameter *seqdetect* is turned off.
3. AVGPARTITION_CLUSTERRATIO, AVGPARTITION_CLUSTERFACTOR, and AVGPARTITION_PAGE_FETCH_PAIRS measure the degree of clustering within a single data partition (local clustering). CLUSTERRATIO, CLUSTERFACTOR, and PAGE_FETCH_PAIRS measure the degree of clustering in the entire table (global clustering). Global clustering and local clustering values can diverge significantly if the table partitioning key is not a prefix of the index key, or when the table partitioning key and the index key are logically independent of each other.
4. The DEFINER column is included for backwards compatibility. See OWNER.
5. The XPTH, XRGN, and XVIP indexes are not recognized by any application programming interface that returns index metadata.

SYSCAT.INDEXEXPLOITRULES

Each row represents an index exploitation rule.

Table 213. SYSCAT.INDEXEXPLOITRULES Catalog View

Column Name	Data Type	Nullable	Description
FUNCID	INTEGER		Identifier for the function.
SPECID	SMALLINT		Number of the predicate specification.
IESCHEMA	VARCHAR (128)		Schema name of the index extension.
IENAME	VARCHAR (128)		Unqualified name of the index extension.
RULEID	SMALLINT		Identifier for the exploitation rule.
SEARCHMETHODID	SMALLINT		Identifier for the search method in the specific index extension.
SEARCHKEY	VARCHAR (640)		Key used to exploit the index.
SEARCHARGUMENT	VARCHAR (2700)		Search arguments used to exploit the index.

Table 213. SYSCAT.INDEXEXPLOITRULES Catalog View (continued)

Column Name	Data Type	Nullable	Description
EXACT	CHAR (1)		• N = Index lookup is not exact in terms of predicate evaluation • Y = Index lookup is exact in terms of predicate evaluation

SYSCAT.INDEXEXTENSIONDEP

Each row represents a dependency of an index extension on some other object. The index extension depends on the object of type BTYPE of name BNAME, so a change to the object affects the index extension.

Table 214. SYSCAT.INDEXEXTENSIONDEP Catalog View

Column Name	Data Type	Nullable	Description
IESCHEMA	VARCHAR (128)		Schema name of the index extension.
IENAME	VARCHAR (128)		Unqualified name of the index extension.
BTYPE	CHAR (1)		Type of object on which there is a dependency. Possible values are: • A = Table alias • B = Trigger • F = Routine • G = Global temporary table • H = Hierachy table • K = Package • L = Detached table • N = Nickname • O = Privilege dependency on all subtables or subviews in a table or view hierarchy • Q = Sequence • R = User-defined data type • S = Materialized query table • T = Table (not typed) • U = Typed table • V = View (not typed) • W = Typed view • X = Index extension • Z = XSR object • q = Sequence alias • u = Module alias • v = Global variable • * = Anchored to the row of a base table
BSCHEMA	VARCHAR (128)		Schema name of the object on which there is a dependency.
BMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the object on which a dependency belongs. The null value if not a module object.

Table 214. SYSCAT.INDEXEXTENSIONDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
BNAME	VARCHAR (128)		Unqualified name of the object on which there is a dependency. For routines (BTYPE = 'F'), this is the specific name.
BMODULEID	INTEGER	Y	Identifier for the module of the object on which there is a dependency.
TABAUTH	SMALLINT	Y	If BTYPE = 'O', 'S', 'T', 'U', 'V', 'W', or 'v', encodes the privileges on the table or view that are required by the dependent index extension; the null value otherwise.

SYSCAT.INDEXEXTENSIONMETHODS

Each row represents a search method. An index extension can contain more than one search method.

Table 215. SYSCAT.INDEXEXTENSIONMETHODS Catalog View

Column Name	Data Type	Nullable	Description
METHODNAME	VARCHAR (128)		Name of the search method.
METHODID	SMALLINT		Number of the method in the index extension.
IESCHEMA	VARCHAR (128)		Schema name of the index extension on which this method is defined.
IENAME	VARCHAR (128)		Unqualified name of the index extension on which this method is defined.
RANGEFUNCSCHEMA	VARCHAR (128)		Schema name of the range-through function.
RANGEFUNCNAME	VARCHAR (128)		Unqualified name of the range-through function.
RANGESPECIFICNAME	VARCHAR (128)		Function-specific name of the range-through function.
FILTERFUNCSCHEMA	VARCHAR (128)	Y	Schema name of the filter function.
FILTERFUNCNAME	VARCHAR (128)	Y	Unqualified name of the filter function.
FILTERSPECIFICNAME	VARCHAR (128)	Y	Function-specific name of the filter function.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.INDEXEXTENSIONPARMS

Each row represents an index extension instance parameter or source key column.

Table 216. SYSCAT.INDEXEXTENSIONPARMS Catalog View

Column Name	Data Type	Nullable	Description
IESCHEMA	VARCHAR (128)		Schema name of the index extension.
IENAME	VARCHAR (128)		Unqualified name of the index extension.
ORDINAL	SMALLINT		Sequence number of the parameter or key column.
PARMNAME	VARCHAR (128)		Name of the parameter or key column.

Table 216. SYSCAT.INDEXEXTENSIONPARMS Catalog View (continued)

Column Name	Data Type	Nullable	Description
TYPESCHEMA	VARCHAR (128)		Schema name of the data type of the parameter or key column.
TYPENAME	VARCHAR (128)		Unqualified name of the data type of the parameter or key column.
LENGTH	INTEGER		Data type length of the parameter or key column.
SCALE	SMALLINT		Data type scale of the parameter or key column; 0 if not applicable.
PARMTYPE	CHAR (1)		• K = Source key column • P = Index extension instance parameter
CODEPAGE	SMALLINT		Code page of the index extension instance parameter; 0 if not a string type.
COLLATIONSCHEMA	VARCHAR (128)	Y	For string types, the schema name of the collation for the parameter; the null value otherwise.
COLLATIONNAME	VARCHAR (128)	Y	For string types, the unqualified name of the collation for the parameter; the null value otherwise.

SYSCAT.INDEXEXTENSIONS

Each row represents an index extension.

Table 217. SYSCAT.INDEXEXTENSIONS Catalog View

Column Name	Data Type	Nullable	Description
IESCHEMA	VARCHAR (128)		Schema name of the index extension.
IENAME	VARCHAR (128)		Unqualified name of the index extension.
OWNER	VARCHAR (128)		Authorization ID of the owner of the index extension.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
CREATE_TIME	TIMESTAMP		Time at which the index extension was defined.
KEYGENFUNCSCHEMA	VARCHAR (128)		Schema name of the key generation function.
KEYGENFUNCNAME	VARCHAR (128)		Unqualified name of the key generation function.
KEYGENSPECIFICNAME	VARCHAR (128)		Name of the key generation function instance (might be system-generated).
TEXT	CLOB (2M)		Full text of the CREATE INDEX EXTENSION statement.
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the index extension.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.INDEXOPTIONS

Each row represents an index-specific option value.

Table 218. SYSCAT.INDEXOPTIONS Catalog View

Column Name	Data Type	Nullable	Description
INDSCHEMA	VARCHAR (128)		Schema name of the index.
INDNAME	VARCHAR (128)		Unqualified name of the index.
OPTION	VARCHAR (128)		Name of the index option.
SETTING	VARCHAR (2048)		Value of the index option.

SYSCAT.INDEXPARTITIONS

Each row represents a partitioned index piece located on one data partition. Note:

- The index partition statistics represent one database partition if the table is created on multiple database partitions.

Table 219. SYSCAT.INDEXPARTITIONS Catalog View

Column Name	Data Type	Nullable	Description
INDSCHEMA	VARCHAR (128)		Schema name of the index.
INDNAME	VARCHAR (128)		Unqualified name of the index.
TABSCHEMA	VARCHAR (128)		Schema name of the table or nickname on which the index is defined.
TABNAME	VARCHAR (128)		Unqualified name of the table or nickname on which the index is defined.
IID	SMALLINT		Identifier for the index.
INDPARTITIONTBSPACEID	INTEGER		Identifier for the index partition table space.
INDPARTITIONOBJECTID	INTEGER		Identifier for the index partition object.
DATAPARTITIONID	INTEGER		This corresponds to the DATAPARTITIONID found in the SYSCAT.DATAPARTITIONS view.
INDCARD	BIGINT		Cardinality of the index partition. This might be different from the cardinality of the corresponding data partition for partitioned indexes that do not have a one-to-one relationship between the data partition rows and the index entries.
NLEAF	BIGINT		Number of leaf pages in the index partition; -1 if statistics are not collected.
NUM_EMPTY_LEAFS	BIGINT		Total number of index leaf pages in the index partition that have all of their row identifiers (RIDs) or block identifiers (BIDs) marked deleted.
NUMRIDS	BIGINT		Total number of row identifiers (RIDs) or block identifiers (BIDs) in the index partition; -1 if not known.

Table 219. SYSCAT.INDEXPARTITIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
NUMRIDS_DELETED	BIGINT		Total number of row identifiers (RIDs) or block identifiers (BIDs) in the index partition that are marked deleted, excluding those identifiers on leaf pages on which all the identifiers are marked deleted.
FULLKEYCARD	BIGINT		Number of distinct full-key values in the index partition; -1 if statistics are not collected.
NLEVELS	SMALLINT		Number of index levels in the index partition; -1 if statistics are not collected.
CLUSTERRATIO	SMALLINT		Degree of data clustering with the index partition; -1 in either of the following situations: • Statistics are not collected • Detailed index statistics are collected. In this situation, CLUSTERFACTOR will be used instead.
CLUSTERFACTOR	DOUBLE		Finer measurement of the degree of clustering; -1 if statistics are not collected.
FIRSTKEYCARD	BIGINT		Number of distinct first-key values; -1 if statistics are not collected.
FIRST2KEYCARD	BIGINT		Number of distinct keys using the first two columns of the index key; -1 if statistics are not collected, or if not applicable.
FIRST3KEYCARD	BIGINT		Number of distinct keys using the first three columns of the index key; -1 if statistics are not collected, or if not applicable.
FIRST4KEYCARD	BIGINT		Number of distinct keys using the first four columns of the index key; -1 if statistics are not collected, or if not applicable.
AVGLEAFKEYSIZE	INTEGER		Average index key size for keys on leaf pages in the index partition; -1 if statistics are not collected.
AVGNLEAFKEYSIZE	INTEGER		Average index key size for keys on non-leaf pages in the index partition; -1 if statistics are not collected.
PCTFREE	SMALLINT		Percentage of each index page to be reserved during the initial building of the index partition. This space is available for data insertions after the index partition has been built.
PAGE_FETCH_PAIRS	VARCHAR (520)		A list of pairs of integers, represented in character form. Each pair represents the number of pages in a hypothetical buffer, and the number of page fetches required to scan the data partition with this index using that hypothetical buffer. Zero-length string if not data is available.

Table 219. SYSCAT.INDEXPARTITIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
SEQUENTIAL_PAGES	BIGINT		Number of leaf pages located on disk in index key order with few or no large gaps between them; -1 if statistics are not collected.
DENSITY	INTEGER		Ratio of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index partition, expressed as a percent (integer between 0 and 100); -1 if statistics are not collected.
AVERAGE_SEQUENCE_GAP	DOUBLE		Gap between index page sequences within the index partition. Detected through a scan of index leaf pages, each gap represents the average number of index pages that must be randomly fetched between sequences of index pages; -1 if not known.
AVERAGE_SEQUENCE_ FETCH_GAP	DOUBLE		Gap between table page sequences when fetching using the index partition. Detected through a scan of index leaf pages, each gap represents the average number of data partition pages that must be randomly fetched between sequences of data partition pages; -1 if not known.
AVERAGE_SEQUENCE_ PAGES	DOUBLE		Average number of index pages that are accessible in sequence (that is, the number of index pages that the prefetchers would detect as being in sequence); -1 if not known.
AVERAGE_SEQUENCE_ FETCH_PAGES	DOUBLE		Average number of data partition pages that are accessible in sequence (that is, the number of data partition pages that the prefetchers would detect as being in sequence) when fetching using the index; -1 if not known.
AVERAGE_RANDOM_PAGES	DOUBLE		Average number of random data partition pages between sequential page accesses; -1 if not known.
AVERAGE_RANDOM_FETCH_ PAGES	DOUBLE		Average number of random data partition pages between sequential page accesses when fetching using the index partition; -1 if not known.
STATS_TIME	TIMESTAMP	Y	Last time that any change was made to the recorded statistics for this index partition. The null value if no statistics are available.
COMPRESSION	CHAR (1)		Specifies whether index compression is activated • N = Not activated • Y = Activated
PCTPAGESSAVED	SMALLINT		Approximate percentage of pages saved in the index as a result of index compression. -1 if statistics are not collected.

SYSCAT.INDEXXMLPATTERNS

Each row represents a pattern clause in an index over an XML column.

Table 220. SYSCAT.INDEXXMLPATTERNS Catalog View

Column Name	Data Type	Nullable	Description
INDSCHEMA	VARCHAR (128)		Schema name of the logical index.
INDNAME	VARCHAR (128)		Unqualified name of the logical index.
PINDNAME	VARCHAR (128)		Unqualified name of the physical index.
PINDID	SMALLINT		Identifier for the physical index.
TYPEMODEL	CHAR (1)		• Q = SQL DATA TYPE (Ignore invalid values) • R = SQL DATA TYPE (Reject invalid values)
DATATYPE	VARCHAR (128)		Name of the data type.
HASHED	CHAR (1)		Indicates whether or not the value is hashed. • N = Not hashed • Y = Hashed
LENGTH	SMALLINT		VARCHAR (*n*) length; 0 otherwise.
PATTERNID	SMALLINT		Identifier for the pattern.
PATTERN	CLOB (2M)	Y	Definition of the pattern.

Note:

1. When indexes over XML columns are created, logical indexes that utilize XML pattern information are created, resulting in the creation of physical B-tree indexes with DB2-generated key columns to support the logical indexes. A physical index is created to support the data type that is specified in the xmltype-clause of the CREATE INDEX statement.

SYSCAT.INVALIDOBJECTS

Each row represents an invalid object.

Table 221. SYSCAT.INVALIDOBJECTS Catalog View

Column Name	Data Type	Nullable	Description
OBJECTSCHEMA	VARCHAR (128)		Schema name of the object being created or revalidated.
OBJECTMODULENAME	VARCHAR (128)	Y	Unqualified name of the module to which the object being created or revalidated belongs. The null value if the object does not belong to a module.
OBJECTNAME	VARCHAR (128)		Unqualified name of the object being created or revalidated. For routines (OBJECTTYPE = 'F'), this is the specific name.
ROUTINENAME	VARCHAR (128)	Y	Unqualified name of the routine.

Table 221. SYSCAT.INVALIDOBJECTS Catalog View (continued)

Column Name	Data Type	Nullable	Description
OBJECTTYPE	CHAR (1)		Type of the object being created or revalidated. Possible values are: • B = Trigger • F = Routine • R = User-defined data type • V = View • v = Global variable
SQLCODE	INTEGER	Y	SQLCODE returned in CREATE with errors or revalidation. The null value if the object has never been revalidated.
SQLSTATE	CHAR (5)	Y	SQLSTATE returned in CREATE with errors or revalidation. The null value if the object has never been revalidated.
ERRORMESSAGE	VARCHAR(70)	Y	Short text for the message associated with SQLCODE. The null value if the object has never been revalidated.
LINENUMBER	INTEGER	Y	Line number where the error occurred in compiled objects. The null value if the object is not a compiled object.
INVALIDATE_TIME	TIMESTAMP		Time at which the object was last invalidated.
LAST_REGEN_TIME	TIMESTAMP	Y	Time at which the object was last revalidated. The null value if the object has never been revalidated.

SYSCAT.KEYCOLUSE

Each row represents a column that participates in a key defined by a unique, primary key, or foreign key constraint.

Table 222. SYSCAT.KEYCOLUSE Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR (128)		Name of the constraint.
TABSCHEMA	VARCHAR (128)		Schema name of the table containing the column.
TABNAME	VARCHAR (128)		Unqualified name of the table containing the column.
COLNAME	VARCHAR (128)		Name of the column.
COLSEQ	SMALLINT		Numeric position of the column in the key for the constraint (initial position is 1). If a constraint uses an existing index, this value is the numeric position of the column in the index.

SYSCAT.MODULEAUTH

Each row represents a user, group, or role that has been granted a privilege on a module.

Table 223. SYSCAT.MODULEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of a privilege
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of a privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
MODULEID	INTEGER		Identifier for the module to which this privilege applies.
MODULESCHEMA	VARCHAR (128)		Schema name of the module to which this privilege applies.
MODULENAME	VARCHAR (128)		Unqualified name of the module to which this privilege applies.
EXECUTEAUTH	CHAR (1)		Privilege to execute objects in the identified module. • G = Held and grantable • N = Not held • Y = Held

SYSCAT.MODULEOBJECTS

Each row represents a function, procedure, global variable, condition, or user-defined type that belongs to a module.

Table 224. SYSCAT.MODULEOBJECTS Catalog View

Column Name	Data Type	Nullable	Description
OBJECTSCHEMA	VARCHAR(128)	N	Schema name of the module.
OBJECTMODULENAME	VARCHAR(128)	N	Unqualified name of the module to which the object belongs.
OBJECTNAME	VARCHAR(128)	N	Unqualified name of the object.
OBJECTTYPE	VARCHAR(9)	N	• CONDITION = The object is a condition • FUNCTION = The object is a function • PROCEDURE = The object is a procedure • TYPE = The object is a data type • VARIABLE = The object is a variable
PUBLISHED	CHAR(1)	N	Indicates whether the object can be referenced outside its module. • N = The object is not published • Y = The object is published
SPECIFICNAME	VARCHAR(128)	N	Routine specific name if OBJECTTYPE is 'FUNCTION', 'METHOD' or 'PROCEDURE'; the null value otherwise.

Table 224. SYSCAT.MODULEOBJECTS Catalog View (continued)

Column Name	Data Type	Nullable	Description
USERDEFINED	CHAR(1)	N	Indicates whether the object is generated by the system or defined by a user. • N = The object is system generated • Y = The object is defined by a user

SYSCAT.MODULES

Each row represents a module.

Table 225. SYSCAT.MODULES Catalog View

Column Name	Data Type	Nullable	Description
MODULESCHEMA	VARCHAR (128)		Schema name of the module.
MODULENAME	VARCHAR (128)		Unqualified name of the module.
MODULEID	INTEGER		Identifier for the module.
DIALECT	VARCHAR(10)		The source dialect of the SQL module. Possible values are: • DB2 SQL PL • PL/SQL • Blank = Not applicable for an alias
OWNER	VARCHAR (128)		Authorization ID of the owner of the module.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
MODULETYPE	CHAR (1)		Type of module. • A = Alias • M = Module • P = PL/SQL package
BASE_MODULESCHEMA	VARCHAR (128)	Y	If MODULETYPE is 'A', contains the schema name of the module or alias that is referenced by this alias; the null value otherwise.
BASE_MODULENAME	VARCHAR (128)	Y	If MODULETYPE is 'A', contains the unqualified name of the module or alias that is referenced by this alias; the null value otherwise.
CREATE_TIME	TIMESTAMP		Time at which the module was created.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.NAMEMAPPINGS

Each row represents the mapping between a ″logical″ object (typed table or view and its columns and indexes, including inherited columns) and the corresponding ″implementation″ object (hierarchy table or hierarchy view and its columns and indexes) that implements the logical object.

Table 226. SYSCAT.NAMEMAPPINGS Catalog View

Column Name	Data Type	Nullable	Description
TYPE	CHAR (1)		• C = Column • I = Index • U = Typed table
LOGICAL_SCHEMA	VARCHAR (128)		Schema name of the logical object.
LOGICAL_NAME	VARCHAR (128)		Unqualified name of the logical object.
LOGICAL_COLNAME	VARCHAR (128)	Y	Name of the logical column if TYPE = 'C'; null value otherwise.
IMPL_SCHEMA	VARCHAR (128)		Schema name of the implementation object that implements the logical object.
IMPL_NAME	VARCHAR (128)		Unqualified name of the implementation object that implements the logical object.
IMPL_COLNAME	VARCHAR (128)	Y	Name of the implementation column if TYPE = 'C'; null value otherwise.

SYSCAT.NICKNAMES

Each row represents a nickname.

Table 227. SYSCAT.NICKNAMES Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the nickname.
TABNAME	VARCHAR (128)		Unqualified name of the nickname.
OWNER	VARCHAR (128)		Authorization ID of the owner of the table, view, alias, or nickname.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
STATUS	CHAR (1)		Status of the object. • C = Set integrity pending • N = Normal • X = Inoperative
CREATE_TIME	TIMESTAMP		Time at which the object was created.
STATS_TIME	TIMESTAMP	Y	Time at which any change was last made to recorded statistics for this object. The null value if statistics are not collected.
COLCOUNT	SMALLINT		Number of columns, including inherited columns (if any).
TABLEID	SMALLINT		Internal logical object identifier.
TBSPACEID	SMALLINT		Internal logical identifier for the primary table space for this object.
CARD	BIGINT		Total number of rows in the table; -1 if statistics are not collected.
NPAGES	BIGINT		Total number of pages on which the rows of the nickname exist; -1 if statistics are not gathered.

Table 227. SYSCAT.NICKNAMES Catalog View (continued)

Column Name	Data Type	Nullable	Description
FPAGES	BIGINT		Total number of pages; -1 if statistics are not gathered.
OVERFLOW	BIGINT		Total number of overflow records; -1 if statistics are not gathered.
PARENTS	SMALLINT	Y	Number of parent tables for this object; that is, the number of referential constraints in which this object is a dependent.
CHILDREN	SMALLINT	Y	Number of dependent tables for this object; that is, the number of referential constraints in which this object is a parent.
SELFREFS	SMALLINT	Y	Number of self-referencing referential constraints for this object; that is, the number of referential constraints in which this object is both a parent and a dependent.
KEYCOLUMNS	SMALLINT	Y	Number of columns in the primary key.
KEYINDEXID	SMALLINT	Y	Index identifier for the primary key index; 0 or the null value if there is no primary key.
KEYUNIQUE	SMALLINT		Number of unique key constraints (other than the primary key constraint) defined on this object.
CHECKCOUNT	SMALLINT		Number of check constraints defined on this object.
DATACAPTURE	CHAR (1)		• L = Nickname participates in data replication, including replication of LONG VARCHAR and LONG VARGRAPHIC columns • N = Nickname does not participate in data replication • Y = Nickname participates in data replication

Table 227. SYSCAT.NICKNAMES Catalog View (continued)

Column Name	Data Type	Nullable	Description
CONST_CHECKED	CHAR (32)		• Byte 1 represents foreign key constraint. • Byte 2 represents check constraint. • Byte 5 represents materialized query table. • Byte 6 represents generated column. • Byte 7 represents staging table. • Byte 8 represents data partitioning constraint. • Other bytes are reserved for future use. Possible values are: • F = In byte 5, the materialized query table cannot be refreshed incrementally. In byte 7, the content of the staging table is incomplete and cannot be used for incremental refresh of the associated materialized query table. • N = Not checked • U = Checked by user • W = Was in 'U' state when the table was placed in set integrity pending state • Y = Checked by system
PARTITION_MODE	CHAR (1)		Reserved for future use.
STATISTICS_PROFILE	CLOB (10M)	Y	RUNSTATS command used to register a statistical profile for the object.
ACCESS_MODE	CHAR (1)		Access restriction state of the object. These states only apply to objects that are in set integrity pending state or to objects that were processed by a SET INTEGRITY statement. Possible values are: • D = No data movement • F = Full access • N = No access • R = Read-only access
CODEPAGE	SMALLINT		Code page of the object. This is the default code page used for all character columns, triggers, check constraints, and expression-generated columns.
REMOTE_TABLE	VARCHAR (128)	Y	Unqualified name of the specific data source object (such as a table or a view) for which the nickname was created.
REMOTE_SCHEMA	VARCHAR (128)	Y	Schema name of the specific data source object (such as a table or a view) for which the nickname was created.
SERVERNAME	VARCHAR (128)	Y	Name of the data source that contains the table or view for which the nickname was created.

Table 227. SYSCAT.NICKNAMES Catalog View (continued)

Column Name	Data Type	Nullable	Description
REMOTE_TYPE	CHAR (1)	Y	Type of object at the data source. • A = Alias • N = Nickname • S = Materialized query table • T = Table (untyped) • V = View (untyped)
CACHINGALLOWED	VARCHAR (1)		• N = Caching is not allowed • Y = Caching is allowed
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the table, view, alias, or nickname.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.PACKAGES

Each row represents a package that has been created by binding an application program.

Table 228. SYSCAT.PACKAGES Catalog View

Column Name	Data Type	Nullable	Description
PKGSCHEMA	VARCHAR (128)		Schema name of the package.
PKGNAME	VARCHAR (128)		Unqualified name of the package.
BOUNDBY	VARCHAR (128)		Authorization ID of the binder and owner of the package.
BOUNDBYTYPE	CHAR (1)		• U = The binder and owner is an individual user
OWNER	VARCHAR (128)		Authorization ID of the binder and owner of the package.
OWNERTYPE	CHAR (1)		• U = The binder and owner is an individual user
DEFAULT_SCHEMA	VARCHAR (128)		Default schema name used for unqualified names in static SQL statements.
VALID[1]	CHAR (1)		• N = Needs rebinding • V = Validate at run time • X = Package is inoperative because some function instance on which it depends has been dropped; explicit rebind is needed • Y = Valid
UNIQUE_ID	CHAR (8) FOR BIT DATA		Identifier for a specific package when multiple packages having the same name exist.
TOTAL_SECT	SMALLINT		Number of sections in the package.
FORMAT	CHAR (1)		Date and time format associated with the package. • 0 = Format associated with the territory code of the client • 1 = USA • 2 = EUR • 3 = ISO • 4 = JIS • 5 = LOCAL

Table 228. SYSCAT.PACKAGES Catalog View (continued)

Column Name	Data Type	Nullable	Description
ISOLATION	CHAR (2)	Y	Isolation level. • CS = Cursor Stability • RR = Repeatable Read • RS = Read Stability • UR = Uncommitted Read
CONCURRENTACCESSRESOLUTION	CHAR (1)	Y	The value of the CONCURRENTACCESSRESOLUTION bind option: • U = USE CURRENTLY COMMITTED • W = WAIT FOR OUTCOME • Blank = Not specified
BLOCKING	CHAR (1)	Y	Cursor blocking option. • B = Block all cursors • N = No blocking • U = Block unambiguous cursors
INSERT_BUF	CHAR (1)		Setting of the INSERT bind option (applies to partitioned database systems). • N = Inserts are not buffered • Y = Inserts are buffered at the coordinator database partition to minimize traffic among database partitions
LANG_LEVEL	CHAR (1)	Y	Setting of the LANGLEVEL bind option. • 0 = SAA1 • 1 = MIA • 2 = SQL92E
FUNC_PATH	CLOB (2K)		SQL path in effect when the package was bound.
QUERYOPT	INTEGER		Optimization class under which this package was bound. Used for rebind operations.
EXPLAIN_LEVEL	CHAR (1)		Indicates whether Explain was requested using the EXPLAIN or EXPLSNAP bind option. • P = Package selection level • Blank = No Explain requested
EXPLAIN_MODE	CHAR (1)		Value of the EXPLAIN bind option. • A = ALL • N = No • R = REOPT • Y = Yes
EXPLAIN_SNAPSHOT	CHAR (1)		Value of the EXPLSNAP bind option. • A = ALL • N = No • R = REOPT • Y = Yes
SQLWARN	CHAR (1)		Indicates whether or not positive SQLCODEs resulting from dynamic SQL statements are returned to the application. • N = No, they are suppressed • Y = Yes
SQLMATHWARN	CHAR (1)		Value of the *dft_sqlmathwarn* database configuration parameter at bind time. Indicates whether arithmetic and retrieval conversion errors return warnings and null values (indicator -2), allowing query processing to continue whenever possible. • N = No, errors are returned • Y = Yes, warnings are returned
CREATE_TIME	TIMESTAMP		Time at which the package was first bound.
EXPLICIT_BIND_TIME	TIMESTAMP		Time at which this package was last changed by: • BIND • REBIND (explicit)

Table 228. SYSCAT.PACKAGES Catalog View (continued)

Column Name	Data Type	Nullable	Description
LAST_BIND_TIME	TIMESTAMP		Time at which the package was last changed by: • BIND • REBIND (explicit) • REBIND (implicit)
ALTER_TIME	TIMESTAMP		Time at which this package was last changed by: • BIND • REBIND (explicit) • REBIND (implicit) • ALTER PACKAGE
CODEPAGE	SMALLINT		Application code page at bind time; -1 if not known.
COLLATIONSCHEMA	VARCHAR (128)		Schema name of the collation for the package.
COLLATIONNAME	VARCHAR (128)		Unqualified name of the collation for the package.
COLLATIONSCHEMA_ORDERBY	VARCHAR (128)		Schema name of the collation for ORDER BY clauses in the package.
COLLATIONNAME_ORDERBY	VARCHAR (128)		Unqualified name of the collation for ORDER BY clauses in the package.
DEGREE	CHAR (5)		Degree of intra-partition parallelism that was specified when the package was bound. • 1 = No parallelism • 2-32767 = User-specified limit • ANY = Degree determined by the system (no limit specified)
MULTINODE_PLANS	CHAR (1)		• N = Package was not bound in a partitioned database environment • Y = Package was bound in a partitioned database environment
INTRA_PARALLEL	CHAR (1)		Use of intra-partition parallelism by static SQL statements within the package. • F = One or more static SQL statements in this package can use intra-partition parallelism; this parallelism has been disabled for use on a system that is not configured for intra-partition parallelism • N = No static SQL statement uses intra-partition parallelism • Y = One or more static SQL statements in the package use intra-partition parallelism
VALIDATE	CHAR (1)		Indicates whether validity checking can be deferred until run time. • B = All checking must be performed at bind time • R = Validation of tables, views, and privileges that do not exist at bind time is performed at run time

Table 228. SYSCAT.PACKAGES Catalog View (continued)

Column Name	Data Type	Nullable	Description
DYNAMICRULES	CHAR (1)		• B = BIND; dynamic SQL statements are executed with DYNAMICRULES BIND behavior • D = DEFINERBIND; when the package is run within a routine context, dynamic SQL statements in the package are executed with DEFINE behavior; when the package is not run within a routine context, dynamic SQL statements in the package are executed with BIND behavior • E = DEFINERRUN; when the package is run within a routine context, dynamic SQL statements in the package are executed with DEFINE behavior; when the package is not run within a routine context, dynamic SQL statements in the package are executed with RUN behavior • H = INVOKEBIND; when the package is run within a routine context, dynamic SQL statements in the package are executed with INVOKE behavior; when the package is not run within a routine context, dynamic SQL statements in the package are executed with BIND behavior • I = INVOKERUN; when the package is run within a routine context, dynamic SQL statements in the package are executed with INVOKE behavior; when the package is not run within a routine context, dynamic SQL statements in the package are executed with RUN behavior • R = RUN; dynamic SQL statements are executed with RUN behavior; this is the default
SQLERROR	CHAR (1)		SQLERROR option on the most recent subcommand that bound or rebound the package. • C = CONTINUE; creates a package, even if errors occur while binding SQL statements • N = NOPACKAGE; does not create a package or a bind file if an error occurs
REFRESHAGE	DECIMAL (20,6)		Timestamp duration indicating the maximum length of time between execution of a REFRESH TABLE statement for a materialized query table (MQT) and when that MQT is used in place of a base table.
FEDERATED	CHAR (1)		• N = FEDERATED bind or prep option is turned off • U = FEDERATED bind or prep option was not specified • Y = FEDERATED bind or prep option is turned on
TRANSFORMGROUP	VARCHAR (1024)	Y	Value of the TRANSFORM GROUP bind option; the null value if a transform group is not specified.
REOPTVAR	CHAR (1)		Indicates whether the access path is determined again at execution time using input variable values. • A = Access path is reoptimized for every OPEN or EXECUTE request • N = Access path is determined at bind time • O = Access path is reoptimized only at the first OPEN or EXECUTE request; it is subsequently cached
OS_PTR_SIZE	INTEGER		Word size for the platform on which the package was created. • 32 = Package is a 32-bit package • 64 = Package is a 64-bit package
PKGVERSION	VARCHAR (64)		Version identifier for the package.

Table 228. SYSCAT.PACKAGES Catalog View (continued)

Column Name	Data Type	Nullable	Description
STATICREADONLY	CHAR (1)		Indicates whether or not static cursors will be treated as READ ONLY. Possible values are: • N = Static cursors take on the attributes that would normally be generated for the given statement text and the setting of the LANGLEVEL precompile option • Y = Any static cursor that does not contain the FOR UPDATE or the FOR READ ONLY clause is considered READ ONLY
FEDERATED_ASYNCHRONY	INTEGER		Indicates the limit on asynchrony (the number of ATQs in the plan) as a bind option when the package was bound. • 0 = No asynchrony • *n* = User-specified limit (32 767 maximum) • -1 = Degree of asynchrony determined by the system • -2 = Degree of asynchrony not specified For a non-federated system, the value is 0.
ANONBLOCK	CHAR (1)		• N = The package is not associated with an anonymous block • Y = The package is associated with an anonymous block
OPTPROFILESCHEMA	VARCHAR (128)	Y	Value of the optimization profile schema specified as part of the OPTPROFILE bind option.
OPTPROFILENAME	VARCHAR (128)	Y	Value of the optimization profile name specified as part of the OPTPROFILE bind option.
PKGID	BIGINT		Identifier for the package.
DBPARTITIONNUM	SMALLINT		Number of the database partition where the package was bound.
DEFINER[2]	VARCHAR (128)		Authorization ID of the binder and owner of the package.
PKG_CREATE_TIME[3]	TIMESTAMP		Time at which the package was first bound.
APREUSE	CHAR (1)		• N = The query compiler will not attempt to reuse access plans • Y = The access plans in this package should be reused, meaning that at rebind time the query compiler will attempt to choose plans like the ones currently in the package
EXTENDEDINDICATOR	CHAR (1)		• N = Extended indicator variable values are not recognized • Y = Extended indicator variable values are recognized
LASTUSED	DATE		Reserved for future use.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. If a function instance with dependencies is dropped, the package is put into an "inoperative" state, and it must be explicitly rebound. If any other object with dependencies is dropped, the package is put into an "invalid" state, and the system will attempt to rebind the package automatically when it is first referenced.
2. The DEFINER column is included for backwards compatibility. See OWNER.
3. The PKG_CREATE_TIME column is included for backwards compatibility. See CREATE_TIME.

SYSCAT.PARTITIONMAPS

Each row represents a distribution map that is used to distribute the rows of a table among the database partitions in a database partition group, based on hashing the table's distribution

Table 229. SYSCAT.PARTITIONMAPS Catalog View

Column Name	Data Type	Nullable	Description
PMAP_ID	SMALLINT		Identifier for the distribution map.

Table 229. SYSCAT.PARTITIONMAPS Catalog View (continued)

Column Name	Data Type	Nullable	Description
PARTITIONMAP	BLOB (65536)		Distribution map, a vector of 32768 two-byte integers for a multiple partition database partition group. For a single partition database partition group, there is one entry denoting the partition number of the single partition.

SYSCAT.PREDICATESPECS

Each row represents a predicate specification.

Table 230. SYSCAT.PREDICATESPECS Catalog View

Column Name	Data Type	Nullable	Description
FUNCSCHEMA	VARCHAR (128)		Schema name of the function.
FUNCNAME	VARCHAR (128)		Unqualified name of the function.
SPECIFICNAME	VARCHAR (128)		Name of the function instance.
FUNCID	INTEGER		Identifier for the function.
SPECID	SMALLINT		Number of this predicate specification.
CONTEXTOP	CHAR (8)		Comparison operator, one of the built-in relational operators (=, <, >, >=, and so on).
CONTEXTEXP	CLOB (2M)		Constant, or an SQL expression.
FILTERTEXT	CLOB (32K)	Y	Text of the data filter expression.

SYSCAT.REFERENCES

Each row represents a referential integrity (foreign key) constraint.

Table 231. SYSCAT.REFERENCES Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR (128)		Name of the constraint.
TABSCHEMA	VARCHAR (128)		Schema name of the dependent table.
TABNAME	VARCHAR (128)		Unqualified name of the dependent table.
OWNER	VARCHAR (128)		Authorization ID of the owner of the constraint.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
REFKEYNAME	VARCHAR (128)		Name of the parent key.
REFTABSCHEMA	VARCHAR (128)		Schema name of the parent table.
REFTABNAME	VARCHAR (128)		Unqualified name of the parent table.
COLCOUNT	SMALLINT		Number of columns in the foreign key.

Table 231. SYSCAT.REFERENCES Catalog View (continued)

Column Name	Data Type	Nullable	Description
DELETERULE	CHAR (1)		Delete rule. • A = NO ACTION • C = CASCADE • N = SET NULL • R = RESTRICT
UPDATERULE	CHAR (1)		Update rule. • A = NO ACTION • R = RESTRICT
CREATE_TIME	TIMESTAMP		Time at which the constraint was defined.
FK_COLNAMES	VARCHAR (640)		This column is no longer used and will be removed in a future release. Use SYSCAT.KEYCOLUSE for this information.
PK_COLNAMES	VARCHAR (640)		This column is no longer used and will be removed in a future release. Use SYSCAT.KEYCOLUSE for this information.
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the constraint.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.ROLEAUTH

Each row represents a role granted to a user, group, role, or PUBLIC.

Table 232. SYSCAT.ROLEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Authorization ID that granted the role.
GRANTORTYPE	CHAR (1)		• U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Authorization ID to which the role was granted.
GRANTEETYPE	CHAR (1)		• G = The grantee is a group • R = The grantee is a role • U = The grantee is an individual user
ROLENAME	VARCHAR (128)		Name of the role.
ROLEID	INTEGER		Identifier for the role.
ADMIN	CHAR (1)		Privilege to grant or revoke the role to or from others, or to comment on the role. • N = Not held • Y = Held

SYSCAT.ROLES

Each row represents a role.

Table 233. SYSCAT.ROLES Catalog View

Column Name	Data Type	Nullable	Description
ROLENAME	VARCHAR (128)		Name of the role.
ROLEID	INTEGER		Identifier for the role.
CREATE_TIME	TIMESTAMP		Time when the role was created.
AUDITPOLICYID	INTEGER	Y	Identifier for the audit policy.
AUDITPOLICYNAME	VARCHAR (128)	Y	Name of the audit policy.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.ROUTINEDEP

Each row represents a dependency of a routine on some other object. The routine depends on the object of type BTYPE of name BNAME, so a change to the object affects the routine.

Table 234. SYSCAT.ROUTINEDEP Catalog View

Column Name	Data Type	Nullable	Description
ROUTINESCHEMA	VARCHAR (128)		Schema name of the routine that has dependencies on another object.
ROUTINEMODULENAME	VARCHAR (128)	Y	Unqualified name of the module.
SPECIFICNAME	VARCHAR (128)		Specific name of the routine that has dependencies on another object.
ROUTINEMODULEID	INTEGER	Y	Identifier for the module of the object that has dependencies on another object.

Table 234. SYSCAT.ROUTINEDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
BTYPE	CHAR (1)		Type of object on which there is a dependency. Possible values are: • A = Table alias • B = Trigger • F = Routine • G = Global temporary table • H = Hierachy table • K = Package • L = Detached table • N = Nickname • O = Privilege dependency on all subtables or subviews in a table or view hierarchy • Q = Sequence • R = User-defined data type • S = Materialized query table • T = Table (not typed) • U = Typed table • V = View (not typed) • W = Typed view • X = Index extension • Z = XSR object • q = Sequence alias • u = Module alias • v = Global variable • * = Anchored to the row of a base table
BSCHEMA	VARCHAR (128)		Schema name of the object on which there is a dependency.
BMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the object on which a dependency belongs. The null value if not a module object.
BNAME	VARCHAR (128)		Unqualified name of the object on which there is a dependency. For routines (BTYPE = 'F'), this is the specific name.
BMODULEID	INTEGER	Y	Identifier for the module of the object on which there is a dependency.
TABAUTH	SMALLINT	Y	If BTYPE = 'O', 'S', 'T', 'U', 'V', 'W', or 'v', encodes the privileges on the table or view that are required by the dependent routine; the null value otherwise.
ROUTINENAME	VARCHAR (128)		This column is no longer used and will be removed in a future release. See SPECIFICNAME.

SYSCAT.ROUTINEOPTIONS

Each row represents a routine-specific option value.

Table 235. SYSCAT.ROUTINEOPTIONS Catalog View

Column Name	Data Type	Nullable	Description
ROUTINESCHEMA	VARCHAR (128)		Schema name of the routine if ROUTINEMODULEID is null; otherwise schema name of the module to which the routine belongs.
ROUTINENAME	VARCHAR (128)		Unqualified name of the routine.
SPECIFICNAME	VARCHAR (128)		Name of the routine instance (might be system-generated).
OPTION	VARCHAR (128)		Name of the federated routine option.
SETTING	VARCHAR (2048)		Value of the federated routine option.

SYSCAT.ROWFIELDS

Each row represents a field that is defined for a user-defined row data type.

Table 236. SYSCAT.ROWFIELDS Catalog View

Column Name	Data Type	Nullable	Description
TYPESCHEMA	VARCHAR (128)		Schema name of the row data type that includes the field.
TYPEMODULENAME	VARCHAR (128)	Y	Unqualified name of the module to which the row data type belongs. The null value if not a module row data type.
TYPENAME	VARCHAR (128)		Unqualified name of the row data type that includes the field.
FIELDNAME	VARCHAR (128)		Field name.
FIELDTYPESCHEMA	VARCHAR (128)		Schema name of the data type of the field.
FIELDTYPEMODULENAME	VARCHAR (128)	Y	Unqualified name of the module to which the data type of the field belongs. The null value if the field data type is not a module user-defined data type.
FIELDTYPENAME	VARCHAR (128)		Unqualified name of the data type of the field.
ORDINAL	SMALLINT		Position of the field in the definition of the row data type, starting with 0.
LENGTH	INTEGER		Length of the field data type. For decimal types, contains the precision.
SCALE	SMALLINT		For decimal types, contains the scale of the field data type; for timestamp types, contains the timestamp precision of the field data type; 0 otherwise.
CODEPAGE	SMALLINT		For string types, denotes the code page; 0 indicates FOR BIT DATA; 0 for non-string types.
COLLATIONSCHEMA	VARCHAR (128)	Y	For string types, the schema name of the collation for the field; the null value otherwise.
COLLATIONNAME	VARCHAR (128)	Y	For string types, the unqualified name of the collation for the field; the null value otherwise.

SYSCAT.ROUTINEPARMOPTIONS

Each row represents a routine parameter-specific option value.

Table 237. SYSCAT.ROUTINEPARMOPTIONS Catalog View

Column Name	Data Type	Nullable	Description
ROUTINESCHEMA	VARCHAR (128)		Schema name of the routine if ROUTINEMODULEID is null; otherwise schema name of the module to which the routine belongs.
ROUTINENAME	VARCHAR (128)		Unqualified name of the routine.
SPECIFICNAME	VARCHAR (128)		Name of the routine instance (might be system-generated).
ORDINAL	SMALLINT		Position of the parameter within the routine signature.
OPTION	VARCHAR (128)		Name of the federated routine option.
SETTING	VARCHAR (2048)		Value of the federated routine option.

SYSCAT.ROUTINEPARMS

Each row represents a parameter or the result of a routine defined in SYSCAT.ROUTINES.

Table 238. SYSCAT.ROUTINEPARMS Catalog View

Column Name	Data Type	Nullable	Description
ROUTINESCHEMA	VARCHAR (128)	Y	Schema name of the routine if ROUTINEMODULEID is null; otherwise schema name of the module to which the routine belongs.
ROUTINEMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the routine belongs. The null value if not a module routine.
ROUTINENAME	VARCHAR (128)	Y	Unqualified name of the routine.
ROUTINEMODULEID	INTEGER	Y	Identifier for the module to which the routine belongs. The null value if not a module routine.
SPECIFICNAME	VARCHAR (128)	Y	Name of the routine instance (might be system-generated).
PARMNAME	VARCHAR (128)	Y	Name of the parameter or result column, or the null value if no name exists.
ROWTYPE	CHAR (1)		• B = Both input and output parameter • C = Result after casting • O = Output parameter • P = Input parameter • R = Result before casting

Table 238. SYSCAT.ROUTINEPARMS Catalog View (continued)

Column Name	Data Type	Nullable	Description
ORDINAL	SMALLINT		If ROWTYPE = 'B', 'O', or 'P', numerical position of the parameter within the routine signature, starting with 1; if ROWTYPE = 'R' and the routine returns a table, numerical position of a named column in the result table, starting with 1; 0 otherwise.
TYPESCHEMA	VARCHAR (128)	Y	Schema name of the data type if TYPEMODULEID is null; otherwise schema name of the module to which the data type belongs.
TYPEMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the data type of the parameter or result belongs. The null value if not a module data type.
TYPENAME	VARCHAR (128)	Y	Unqualified name of the data type.
LOCATOR	CHAR (1)		• N = Paramater or result is not passed in the form of a locator • Y = Paramater or result is passed in the form of a locator
LENGTH[1]	INTEGER		Length of the parameter or result; 0 if the parameter or result is a user-defined data type.
SCALE[1]	SMALLINT		Scale if the parameter or result data type is DECIMAL; the number of digits of fractional seconds if the parameter or result data type is TIMESTAMP; 0 otherwise.
CODEPAGE	SMALLINT		Code page of this parameter or result; 0 denotes either not applicable, or a parameter or result for character data declared with the FOR BIT DATA attribute.
COLLATIONSCHEMA	VARCHAR (128)	Y	For string types, the schema name of the collation for the parameter; the null value otherwise.
COLLATIONNAME	VARCHAR (128)	Y	For string types, the unqualified name of the collation for the parameter; the null value otherwise.
CAST_FUNCSCHEMA	VARCHAR (128)	Y	Schema name of the function used to cast an argument or a result. Applies to sourced and external functions; the null value otherwise.
CAST_FUNCSPECIFIC	VARCHAR (128)	Y	Unqualified name of the function used to cast an argument or a result. Applies to sourced and external functions; the null value otherwise.
TARGET_TYPESCHEMA	VARCHAR (128)	Y	Schema name of the target type if the type of the parameter or result is REFERENCE. Null value if the type of the parameter or result is not REFERENCE.

Table 238. SYSCAT.ROUTINEPARMS Catalog View (continued)

Column Name	Data Type	Nullable	Description
TARGET_TYPEMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the target type belongs if the type of the parameter or result is REFERENCE. The null value if the type of the parameter or result is not REFERENCE or if the target type is not a module data type.
TARGET_TYPENAME	VARCHAR (128)	Y	Unqualified name of the module to which the target type belongs if the type of the parameter or result is REFERENCE. The null value if the type of the parameter or result is not REFERENCE or if the target type is not a module data type.
SCOPE_TABSCHEMA	VARCHAR (128)	Y	Schema name of the scope (target table) if the parameter type is REFERENCE; null value otherwise.
SCOPE_TABNAME	VARCHAR (128)	Y	Unqualified name of the scope (target table) if the parameter type is REFERENCE; null value otherwise.
TRANSFORMGRPNAME	VARCHAR (128)	Y	Name of the transform group for a structured type parameter or result.
DEFAULT	CLOB (64K)	Y	Expression used to calculate the default value of the parameter. The null value if DEFAULT clause was not specified for the parameter.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. LENGTH and SCALE are set to 0 for sourced functions (functions defined with a reference to another function), because they inherit the length and scale of parameters from their source.

SYSCAT.ROUTINES

Each row represents a user-defined routine (scalar function, table function, sourced function, method, or procedure). Does not include built-in functions.

Table 239. SYSCAT.ROUTINES Catalog View

Column Name	Data Type	Nullable	Description
ROUTINESCHEMA	VARCHAR (128)		Schema name of the routine if ROUTINEMODULEID is null; otherwise schema name of the module to which the routine belongs.
ROUTINEMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the routine belongs. The null value if not a module routine.
ROUTINENAME	VARCHAR (128)		Unqualified name of the routine.
ROUTINETYPE	CHAR (1)		Type of routine. • F = Function • M = Method • P = Procedure
OWNER	VARCHAR (128)		Authorization ID of the owner of the routine.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user

Table 239. SYSCAT.ROUTINES Catalog View (continued)

Column Name	Data Type	Nullable	Description
SPECIFICNAME	VARCHAR (128)		Name of the routine instance (might be system-generated).
ROUTINEID	INTEGER		Identifier for the routine.
ROUTINEMODULEID	INTEGER	Y	Identifier for the module to which the routine belongs. The null value if not a module routine.
RETURN_TYPESCHEMA	VARCHAR (128)	Y	Schema name of the return type for a scalar function or method.
RETURN_TYPEMODULE	VARCHAR (128)	Y	The module name of the return type; the null value if the return type does not belong to any module.
RETURN_TYPENAME	VARCHAR (128)	Y	Unqualified name of the return type for a scalar function or method.
ORIGIN	CHAR (1)		• B = Built-in • E = User-defined, external • M = Template function • F = Federated procedure • Q = SQL-bodied[1] • R = System-generated SQL-bodied routine • S = System-generated • T = System-generated transform function (not directly invokable) • U = User-defined, based on a source
FUNCTIONTYPE	CHAR (1)		• C = Column or aggregate • R = Row • S = Scalar • T = Table • Blank = Procedure
PARM_COUNT	SMALLINT		Number of routine parameters.
LANGUAGE	CHAR (8)		Implementation language for the routine body (or for the source function body, if this function is sourced on another function). Possible values are 'C', 'COBOL', 'JAVA', 'OLE', 'OLEDB', or 'SQL'. Blanks if ORIGIN is not 'E', 'Q', or 'R'.
DIALECT	VARCHAR(10)		The source dialect of the SQL routine body: • DB2 SQL PL • PL/SQL • Blank = Not an SQL routine
SOURCESCHEMA	VARCHAR (128)	Y	If ORIGIN = 'U' and the source function is a user-defined function, contains the schema name of the specific name of the source function. If ORIGIN = 'U' and the source function is a built-in function, contains the value 'SYSIBM'. The null value if ORIGIN is not 'U'.
SOURCESPECIFIC	VARCHAR (128)	Y	If ORIGIN = 'U' and the source function is a user-defined function, contains the unqualified specific name of the source function. If ORIGIN = 'U' and the source function is a built-in function, contains the value 'N/A for built-in'. The null value if ORIGIN is not 'U'.

Table 239. SYSCAT.ROUTINES Catalog View (continued)

Column Name	Data Type	Nullable	Description
PUBLISHED	CHAR (1)		Inidicates whether the module routine can be invoked outside its module. • N = The module routine is not published • Y = The module routine is published • Blank = Not applicable
DETERMINISTIC	CHAR (1)		• N = Results are not deterministic (same parameters might give different results in different routine calls) • Y = Results are deterministic • Blank = ORIGIN is not 'E', 'F', 'Q', or 'R'
EXTERNAL_ACTION	CHAR (1)		• E = Function has external side-effects (therefore, the number of invocations is important) • N = No side-effects • Blank = ORIGIN is not 'E', 'F', 'Q', or 'R'
NULLCALL	CHAR (1)		• N = RETURNS NULL ON NULL INPUT (function result is implicitly the null value if one or more operands are null) • Y = CALLED ON NULL INPUT • Blank = ORIGIN is not 'E', 'Q', or 'R'
CAST_FUNCTION	CHAR (1)		• N = Not a cast function • Y = Cast function • Blank = ROUTINETYPE is not 'F'
ASSIGN_FUNCTION	CHAR (1)		• N = Not an assignment function • Y = Implicit assignment function • Blank = ROUTINETYPE is not 'F'
SCRATCHPAD	CHAR (1)		• N = Routine has no scratchpad • Y = Routine has a scratchpad • Blank = ORIGIN is not 'E' or ROUTINETYPE is 'P'
SCRATCHPAD_LENGTH	SMALLINT		Size (in bytes) of the scratchpad for the routine. • -1 = LANGUAGE is 'OLEDB' and SCRATCHPAD is 'Y' • 0 = SCRATCHPAD is not 'Y'
FINALCALL	CHAR (1)		• N = No final call is made • Y = Final call is made to this routine at the runtime end-of-statement • Blank = ORIGIN is not 'E' or ROUTINETYPE is 'P'
PARALLEL	CHAR (1)		• N = Routine cannot be executed in parallel • Y = Routine can be executed in parallel • Blank = ORIGIN is not 'E'

Table 239. SYSCAT.ROUTINES Catalog View (continued)

Column Name	Data Type	Nullable	Description
PARAMETER_STYLE	CHAR (8)		Parameter style that was declared when the routine was created. Possible values are: • DB2DARI • DB2GENRL • DB2SQL • GENERAL • GNRLNULL • JAVA • SQL • Blanks if ORIGIN is not 'E'
FENCED	CHAR (1)		• N = Not fenced • Y = Fenced • Blank = ORIGIN is not 'E'
SQL_DATA_ACCESS	CHAR (1)		Indicates what type of SQL statements, if any, the database manager should assume is contained in the routine. • C = Contains SQL (simple expressions with no subqueries only) • M = Contains SQL statements that modify data • N = Does not contain SQL statements • R = Contains read-only SQL statements • Blank = ORIGIN is not 'E', 'F', 'Q', or 'R'
DBINFO	CHAR (1)		Indicates whether a DBINFO parameter is passed to an external routine. • N = DBINFO is not passed • Y = DBINFO is passed • Blank = ORIGIN is not 'E'
PROGRAMTYPE	CHAR (1)		Indicates how the external routine is invoked. • M = Main • S = Subroutine • Blank = ORIGIN is 'F'
COMMIT_ON_RETURN	CHAR (1)		Indicates whether the transaction is committed on successful return from this procedure. • N = The unit of work is not committed • Y = The unit of work is committed • Blank = ROUTINETYPE is not 'P'
AUTONOMOUS	CHAR (1)		Indicates whether or not the routine executes autonomously. • N = Routine does not execute autonomously from invoking transaction • Y = Routine executes autonomously from invoking transaction • Blank = ROUTINETYPE is not 'P'
RESULT_SETS	SMALLINT		Estimated maximum number of result sets.
SPEC_REG	CHAR (1)		Indicates the special registers values that are used when the routine is called. • I = Inherited special registers • Blank = PARAMETER_STYLE is 'DB2DARI' or ORIGIN is not 'E', 'Q', or 'R'

Table 239. SYSCAT.ROUTINES Catalog View (continued)

Column Name	Data Type	Nullable	Description
FEDERATED	CHAR (1)		Indicates whether or not federated objects can be accessed from the routine. • Y = Federated objects can be accessed • Blank = ORIGIN is not 'F'
THREADSAFE	CHAR (1)		Indicates whether or not the routine can run in the same process as other routines. • N = Routine is not threadsafe • Y = Routine is threadsafe • Blank = ORIGIN is not 'E'
VALID	CHAR (1)		Applies to LANGUAGE = 'SQL' and routines having parameters with default; blank otherwise. • N = Routine needs rebinding • X = Routine is inoperative and must be recreated • Y = Routine is valid
MODULEROUTINEIMPLEMENTED	CHAR (1)		• N = Module routine body is not implemented • Y = Module routine body is implemented • Blank = ROUTINEMODULENAME is null value
METHODIMPLEMENTED	CHAR (1)		• N = Method body is not implemented • Y = Method body is implemented • Blank = ROUTINETYPE is not 'M' or ROUTINEMODULENAME is not the null value
METHODEFFECT	CHAR (2)		• CN = Constructor method • MU = Mutator method • OB = Observer method • Blanks = Not a system method
TYPE_PRESERVING	CHAR (1)		• N = Return type is the declared return type of the method • Y = Return type is governed by a "type-preserving" parameter; all system-generated mutator methods are type-preserving • Blank = ROUTINETYPE is not 'M'
WITH_FUNC_ACCESS	CHAR (1)		• N = This method cannot be invoked by using functional notation • Y = This method can be invoked by using functional notation; that is, the "WITH FUNCTION ACCESS" attribute is specified • Blank = ROUTINETYPE is not 'M'
OVERRIDDEN_METHODID	INTEGER	Y	Identifier for the overridden method when the OVERRIDING option is specified for a user-defined method. The null value if ROUTINETYPE is not 'M'.
SUBJECT_TYPESCHEMA	VARCHAR (128)	Y	Schema name of the subject type for the user-defined method. The null value if ROUTINETYPE is not 'M'.
SUBJECT_TYPENAME	VARCHAR (128)	Y	Unqualified name of the subject type for the user-defined method. The null value if ROUTINETYPE is not 'M'.

Table 239. SYSCAT.ROUTINES Catalog View (continued)

Column Name	Data Type	Nullable	Description
CLASS	VARCHAR (384)	Y	For LANGUAGE JAVA, CLR, or OLE, this is the class that implements this routine; null value otherwise.
JAR_ID	VARCHAR (128)	Y	For LANGUAGE JAVA, this is the JAR_ID of the installed jar file that implements this routine if a jar file was specified at routine creation time; null value otherwise. For LANGUAGE CLR, this is the assembly file that implements this routine.
JARSCHEMA	VARCHAR (128)	Y	For LANGUAGE JAVA when a JAR_ID is present, this is the schema name of the jar file that implements this routine; null value otherwise.
JAR_SIGNATURE	VARCHAR (2048)	Y	For LANGUAGE JAVA, this is the method signature of this routine's specified Java method. For LANGUAGE CLR, this is a reference field for this CLR routine. Null value otherwise.
CREATE_TIME	TIMESTAMP		Time at which the routine was created.
ALTER_TIME	TIMESTAMP		Time at which the routine was last altered.
FUNC_PATH	CLOB (2K)	Y	SQL path in effect when the routine was defined. The null value if LANGUAGE is not 'SQL'.
QUALIFIER	VARCHAR (128)		Value of the default schema at the time of object definition. Used to complete any unqualified references.
IOS_PER_INVOC	DOUBLE		Estimated number of inputs/outputs (I/Os) per invocation; 0 is the default; -1 if not known.
INSTS_PER_INVOC	DOUBLE		Estimated number of instructions per invocation; 450 is the default; -1 if not known.
IOS_PER_ARGBYTE	DOUBLE		Estimated number of I/Os per input argument byte; 0 is the default; -1 if not known.
INSTS_PER_ARGBYTE	DOUBLE		Estimated number of instructions per input argument byte; 0 is the default; -1 if not known.
PERCENT_ARGBYTES	SMALLINT		Estimated average percent of input argument bytes that the routine will actually read; 100 is the default; -1 if not known.
INITIAL_IOS	DOUBLE		Estimated number of I/Os performed the first time that the routine is invoked; 0 is the default; -1 if not known.
INITIAL_INSTS	DOUBLE		Estimated number of instructions executed the first time the routine is invoked; 0 is the default; -1 if not known.
CARDINALITY	BIGINT		Predicted cardinality of a table function; -1 if not known, or if the routine is not a table function.
SELECTIVITY[2]	DOUBLE		For user-defined predicates; -1 if there are no user-defined predicates.
RESULT_COLS	SMALLINT		For a table function (ROUTINETYPE = 'F' and FUNCTIONTYPE = 'T'), contains the number of columns in the result table; for a procedure (ROUTINETYPE = 'P'), contains 0; contains 1 otherwise.
IMPLEMENTATION	VARCHAR (762)	Y	If ORIGIN = 'E', identifies the path/module/function that implements this function. If ORIGIN = 'U' and the source function is built-in, this column contains the name and signature of the source function. Null value otherwise.
LIB_ID	INTEGER	Y	Reserved for future use.

Table 239. SYSCAT.ROUTINES Catalog View (continued)

Column Name	Data Type	Nullable	Description
TEXT_BODY_OFFSET	INTEGER		If LANGUAGE = 'SQL', the offset to the start of the SQL procedure body in the full text of the CREATE statement; -1 if LANGUAGE is not 'SQL'.
TEXT	CLOB (2M)	Y	If LANGUAGE = 'SQL', the full text of the CREATE FUNCTION, CREATE METHOD, or CREATE PROCEDURE statement; null value otherwise.
NEWSAVEPOINTLEVEL	CHAR (1)		Indicates whether the routine initiates a new savepoint level when it is invoked. • N = A new savepoint level is not initiated when the routine is invoked; the routine uses the existing savepoint level • Y = A new savepoint level is initiated when the routine is invoked • Blank = Not applicable
DEBUG_MODE[3]	VARCHAR (8)		Indicates whether or not the routine can be debugged using the DB2 debugger. • DISALLOW = Routine is not debuggable • ALLOW = Routine is debuggable, and can participate in a client debug session with the DB2 debugger • DISABLE = Routine is not debuggable, and this setting cannot be altered without dropping and recreating the routine • Blank = Routine type is not currently supported by the DB2 debugger
TRACE_LEVEL	VARCHAR (1)	Y	Reserved for future use.
DIAGNOSTIC_LEVEL	VARCHAR (1)	Y	Reserved for future use.
CHECKOUT_USERID	VARCHAR (128)	Y	ID of the user who performed a checkout of the object; the null value if the object is not checked out.
PRECOMPILE_OPTIONS	VARCHAR (1024)	Y	Precompile options specified for the routine.
COMPILE_OPTIONS	VARCHAR (1024)	Y	Compile options specified for the routine.
EXECUTION_CONTROL	CHAR (1)		Execution control mode of a common language runtime (CLR) routine. Possible values are: • N = Network • R = Fileread • S = Safe • U = Unsafe • W = Filewrite • Blank = LANGUAGE is not 'CLR'
CODEPAGE	SMALLINT		Routine code page, which specifies the default code page used for all character parameter types, result types, and local variables within the routine body.
COLLATIONSCHEMA	VARCHAR (128)		Schema name of the collation for the routine.
COLLATIONNAME	VARCHAR (128)		Unqualified name of the collation for the routine.
COLLATIONSCHEMA_ORDERBY	VARCHAR (128)		Schema name of the collation for ORDER BY clauses in the routine.
COLLATIONNAME_ORDERBY	VARCHAR (128)		Unqualified name of the collation for ORDER BY clauses in the routine.

Table 239. SYSCAT.ROUTINES Catalog View (continued)

Column Name	Data Type	Nullable	Description
ENCODING_SCHEME	CHAR (1)		Encoding scheme of the routine, as specified in the PARAMETER CCSID clause. Possible values are: • A = ASCII • U = UNICODE • Blank = PARAMETER CCSID clause was not specified
LAST_REGEN_TIME	TIMESTAMP		Time at which the SQL routine packed descriptor was last regenerated.
INHERITLOCKREQUEST	CHAR (1)		• N = This function or method cannot be invoked in the context of an SQL statement that includes a lock-request-clause as part of a specified isolation-clause • Y = This function or method inherits the isolation level of the invoking statement; it also inherits the specified lock-request-clause • Blank = LANGUAGE is not 'SQL' or ROUTINETYPE is 'P'
DEFINER[4]	VARCHAR (128)		Authorization ID of the owner of the routine.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. For SQL procedures created before Version 8.2 and upgraded to Version 9, 'E' (instead of 'Q').
2. During database upgrade, the SELECTIVITY column will be set to -1 in the packed descriptor and system catalogs for all user-defined routines. For a user-defined predicate, the selectivity in the system catalog will be -1. In this case, the selectivity value used by the optimizer is 0.01.
3. For Java routines, the DEBUG_MODE setting does not indicate whether the Java routine was actually compiled in debug mode, or whether a debug Jar was installed at the server.
4. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.ROUTINESFEDERATED

Each row represents a federated procedure.

Table 240. SYSCAT.ROUTINESFEDERATED Catalog View

Column Name	Data Type	Nullable	Description
ROUTINESCHEMA	VARCHAR (128)		Schema name of the routine if ROUTINEMODULEID is null; otherwise schema name of the module to which the routine belongs.
ROUTINENAME	VARCHAR (128)		Unqualified name of the routine.
ROUTINETYPE	CHAR (1)		Type of routine. • P = Procedure
OWNER	VARCHAR (128)		Authorization ID of the owner of the routine.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
SPECIFICNAME	VARCHAR (128)		Name of the routine instance (might be system-generated).
ROUTINEID	INTEGER		Identifier for the routine.
PARM_COUNT	SMALLINT		Number of routine parameters.

Table 240. SYSCAT.ROUTINESFEDERATED Catalog View (continued)

Column Name	Data Type	Nullable	Description
DETERMINISTIC	CHAR (1)		• N = Results are not deterministic (same parameters might give different results in different routine calls) • Y = Results are deterministic
EXTERNAL_ACTION	CHAR (1)		• E = Routine has external side-effects (therefore, the number of invocations is important) • N = No side-effects
SQL_DATA_ACCESS	CHAR (1)		Indicates what type of SQL statements, if any, the database manager should assume is contained in the routine. • C = Contains SQL (simple expressions with no subqueries only) • M = Contains SQL statements that modify data • N = Does not contain SQL statements • R = Contains read-only SQL statements
COMMIT_ON_RETURN	CHAR (1)		Indicates whether the transaction is committed on successful return from this procedure. • N = The unit of work is not committed • Y = The unit of work is committed • Blank = ROUTINETYPE is not 'P'
RESULT_SETS	SMALLINT		Estimated maximum number of result sets.
CREATE_TIME	TIMESTAMP		Time at which the routine was created.
ALTER_TIME	TIMESTAMP		Time at which the routine was last altered.
QUALIFIER	VARCHAR (128)		Value of the default schema at the time of object definition. Used to complete any unqualified references.
RESULT_COLS	SMALLINT		For a procedure (ROUTINETYPE = 'P'), contains 0; contains 1 otherwise.
CODEPAGE	SMALLINT		Routine code page, which specifies the default code page used for all character parameter types, result types, and local variables within the routine body.
LAST_REGEN_TIME	TIMESTAMP		Time at which the SQL routine packed descriptor was last regenerated.
REMOTE_PROCEDURE	VARCHAR (128)	Y	Unqualified name of the source procedure for which the federated routine was created.
REMOTE_SCHEMA	VARCHAR (128)	Y	Schema name of the source procedure for which the federated routine was created.
SERVERNAME	VARCHAR (128)	Y	Name of the data source that contains the source procedure for which the federated routine was created.
REMOTE_PACKAGE	VARCHAR (128)	Y	Name of the package to which the source procedure belongs (applies only to wrappers for Oracle data sources).

Table 240. SYSCAT.ROUTINESFEDERATED Catalog View (continued)

Column Name	Data Type	Nullable	Description
REMOTE_PROCEDURE_ ALTER_TIME	VARCHAR (128)	Y	Reserved for future use.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.SERVEROPTIONS

Each row represents a server-specific option value.

Table 241. SYSCAT.SERVEROPTIONS Catalog View

Column Name	Data Type	Nullable	Description
WRAPNAME	VARCHAR (128)	Y	Name of the wrapper.
SERVERNAME	VARCHAR (128)	Y	Uppercase name of the server.
SERVERTYPE	VARCHAR (30)	Y	Type of server.
SERVERVERSION	VARCHAR (18)	Y	Server version.
CREATE_TIME	TIMESTAMP		Time at which the entry was created.
OPTION	VARCHAR (128)		Name of the server option.
SETTING	VARCHAR (2048)		Value of the server option.
SERVEROPTIONKEY	VARCHAR (18)		Uniquely identifies a row.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.SERVERS

Each row represents a data source.

Table 242. SYSCAT.SERVERS Catalog View

Column Name	Data Type	Nullable	Description
WRAPNAME	VARCHAR (128)		Name of the wrapper.
SERVERNAME	VARCHAR (128)		Uppercase name of the server.
SERVERTYPE	VARCHAR (30)	Y	Type of server.
SERVERVERSION	VARCHAR (18)	Y	Server version.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.SERVICECLASSES

Each row represents a service class.

Table 243. SYSCAT.SERVICECLASSES Catalog View

Column Name	Data Type	Nullable	Description
SERVICECLASSNAME	VARCHAR (128)		Name of the service class.
PARENTSERVICECLASSNAME	VARCHAR (128)	Y	Service class name of the parent service superclass.
SERVICECLASSID	SMALLINT		Identifier for the service class.

Table 243. SYSCAT.SERVICECLASSES Catalog View (continued)

Column Name	Data Type	Nullable	Description
PARENTID	SMALLINT		Identifier for the parent service class for this service class. 0 if this service class is a super service class.
CREATE_TIME	TIMESTAMP		Time when the service class was created.
ALTER_TIME	TIMESTAMP		Time when the service class was last altered.
ENABLED	CHAR (1)		State of the service class. • N = Disabled • Y = Enabled
AGENTPRIORITY	SMALLINT		Thread priority of the agents in the service class relative to the normal priority of DB2 threads. • -20 to 20 (Linux and UNIX) • -6 to 6 (Windows) • -32768 = not set
PREFETCHPRIORITY	CHAR (1)		Prefetch priority of the agents in the service class. • H = High • L = Low • M = Medium • Blank = not set
BUFFERPOOLPRIORITY	CHAR (1)		Bufferpool priority of the agents in the service class • H = High • L = Low • M = Medium • Blank = Not set
INBOUNDCORRELATOR	VARCHAR (128)	Y	For future use.
OUTBOUNDCORRELATOR	VARCHAR (128)	Y	String used to associate the service class with an operating system workload manager service class.
COLLECTAGGACTDATA	CHAR (1)		Specifies what aggregate activity data should be captured for the service class by the applicable event monitor. • B = Collect base aggregate activity data • E = Collect extended aggregate activity data • N = None
COLLECTAGGREQDATA	CHAR (1)		Specifies what aggregate activity data should be captured for the service class by the applicable event monitor. • B = Collect base aggregate request data • N = None

Table 243. SYSCAT.SERVICECLASSES Catalog View (continued)

Column Name	Data Type	Nullable	Description
COLLECTACTDATA	CHAR (1)		Specifies what activity data should be collected by the applicable event monitor. • D = Activity data with details • N = None • S = Activity data with details and section environment • V = Activity data with details and values • W = Activity data without details • X = Activity data with details, section environment, and values
COLLECTACTPARTITION	CHAR (1)		Specifies where activity data is collected. • C = Database partition of the coordinator of the activity • D = All database partitions
COLLECTREQMETRICS	CHAR (1)		Specifies the monitoring level for requests submitted by a connection that is associated with the service superclass. • B = Collect base request metrics • E = Collect extended request metrics • N = None
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.STATEMENTS

Each row represents an SQL statement in a package.

Table 244. SYSCAT.STATEMENTS Catalog View

Column Name	Data Type	Nullable	Description
PKGSCHEMA	VARCHAR (128)		Schema name of the package.
PKGNAME	VARCHAR (128)		Unqualified name of the package.
STMTNO	INTEGER		Line number of the SQL statement in the source module of the application program.
SECTNO	SMALLINT		Number of the package section containing the SQL statement.
SEQNO	INTEGER		Always 1.
TEXT	CLOB (2M)		Text of the SQL statement.
UNIQUE_ID	CHAR (8) FOR BIT DATA		Identifier for a specific package when multiple packages having the same name exist.
VERSION	VARCHAR (64)	Y	Version identifier for the package.

SYSCAT.TABDEP

Each row represents a dependency of a view or a materialized query table on some other object. The view or materialized query table depends on the object of type

BTYPE of name BNAME, so a change to the object affects the view or materialized query table. Also encodes how privileges on views depend on privileges on underlying tables and views.

Table 245. SYSCAT.TABDEP Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the view or materialized query table.
TABNAME	VARCHAR (128)		Unqualified name of the view or materialized query table.
DTYPE	CHAR (1)		Type of the depending object. • S = Materialized query table • T = Table (staging only) • V = View (untyped) • W = Typed view
OWNER	VARCHAR (128)		Authorization ID of the creator of the view or materialized query table.
OWNERTYPE	CHAR (1)		• U = The owner is an individual user
BTYPE	CHAR (1)		Type of object on which there is a dependency. Possible values are: • A = Table alias • F = Routine • I = Index, if recording dependency on a base table • G = Global temporary table • N = Nickname • O = Privilege dependency on all subtables or subviews in a table or view hierarchy • R = User-defined structured type • S = Materialized query table • T = Table (untyped) • U = Typed table • V = View (untyped) • W = Typed view • Z = XSR object • u = Module alias • v = Global variable
BSCHEMA	VARCHAR (128)		Schema name of the object on which the view or materialized query table depends.
BMODULENAME	VARCHAR (128)	Y	Unqualified name of the module to which the object on which there is a dependency belongs. The null value if not a module object.
BNAME	VARCHAR (128)		Unqualified name of the object on which the view or materialized query table depends.
BMODULEID	INTEGER	Y	Identifier for the module of the object on which the view or materialized query table depends.

Table 245. SYSCAT.TABDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
TABAUTH	SMALLINT	Y	If BTYPE is 'N', 'O', 'S', 'T', 'U', 'V', or 'W', encodes the privileges on the underlying table or view on which this view or materialized query table depends; the null value otherwise.
VARAUTH	SMALLINT	Y	If BTYPE is 'v', encodes the privileges on the underlying global variable on which this view or materialized query table depends; the null value otherwise.
DEFINER[1]	VARCHAR (128)		Authorization ID of the creator of the view or materialized query table.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.TABDETACHEDDEP

Each row represents a detached dependency between a detached dependent table and a detached table.

Table 246. SYSCAT.TABDETACHEDDEP Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the detached table.
TABNAME	VARCHAR (128)		Unqualified name of the detached table.
DEPTABSCHEMA	VARCHAR (128)		Schema name of the detached dependent table.
DEPTABNAME	VARCHAR (128)		Unqualified name of the detached dependent table.

SYSCAT.TABOPTIONS

Each row represents an option that is associated with a remote table.

Table 247. SYSCAT.TABOPTIONS Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of a table, view, alias, or nickname.
TABNAME	VARCHAR (128)		Unqualified name of a table, view, alias, or nickname.
OPTION	VARCHAR (128)		Name of the table option.
SETTING	CLOB (32K)		Value of the table option.

SYSCAT.THRESHOLDS

Each row represents a threshold.

Table 248. SYSCAT.THRESHOLDS Catalog View

Column Name	Data Type	Nullable	Description
THRESHOLDNAME	VARCHAR (128)		Name of the threshold.
THRESHOLDID	INTEGER		Identifier for the threshold.
ORIGIN	CHAR (1)		Origin of the threshold. • U = Threshold was created by a user • W = Threshold was created through a work action set
THRESHOLDCLASS	CHAR (1)		Classification of the threshold. • A = Aggregate threshold • C = Activity threshold
THRESHOLDPREDICATE	VARCHAR (15)		Type of the threshold. Possible values are: • AGGTEMPSPACE • CONCDBC • CONCWCN • CONCWOC • CONNIDLETIME • CPUTIME • CPUTIMEINSC • DBCONN • ESTSQLCOST • ROWSREAD • ROWSREADINSC • ROWSRET • SCCONN • TEMPSPACE • TOTALTIME
THRESHOLDPREDICATEID	SMALLINT		Identifier for the threshold predicate.
DOMAIN	CHAR (2)		Domain of the threshold. • DB = Database • SB = Service subclass • SP = Service superclass • WA = Work action set • WD = Workload definition
DOMAINID	INTEGER		Identifier for the object with which the threshold is associated. This can be a service class, work action or workload unique ID. If this is a database threshold, this value is 0.
ENFORCEMENT	CHAR (1)		Scope of enforcement for the threshold. • D = Database • P = Database partition • W = Workload occurrence
QUEUING	CHAR (1)		• N = The threshold is not queueing • Y = The threshold is queueing
MAXVALUE	BIGINT		Upper bound specified by the threshold.
QUEUESIZE	INTEGER		If QUEUEING is 'Y', the size of the queue. -1 otherwise.

Table 248. SYSCAT.THRESHOLDS Catalog View (continued)

Column Name	Data Type	Nullable	Description
COLLECTACTDATA	CHAR (1)		Specifies what activity data should be collected by the applicable event monitor. • D = Activity data with details • N = None • S = Activity data with details and section environment • V = Activity data with details and values • W = Activity data without details • X = Activity data with details, section environment, and values
COLLECTACTPARTITION	CHAR (1)		Specifies where activity data is collected. • C = Database partition of the coordinator of the activity • D = All database partitions
EXECUTION	CHAR (1)		Indicates whether execution continues, stops, or remaps to a different service subclass after the threshold has been exceeded. • C = Execution continues • R = Execution is remapped • S = Execution stops
REMAPSCID	SMALLINT		Target service subclass ID of the REMAP ACTIVITY action.
VIOLATIONRECORDLOGGED	CHAR (1)		Indicates whether a record is written to the event monitor upon threshold violation. • N = No • Y = Yes
CHECKINTERVAL	INTEGER		The interval, in seconds, in which the threshold condition is checked if THRESHOLDPREDICATE is: • 'CPUTIME' • 'CPUTIMEINSC' • 'ROWSREAD' • 'ROWSREADINSC' Otherwise, -1.
ENABLED	CHAR (1)		• N = This threshold is disabled. • Y = This threshold is enabled.
CREATE_TIME	TIMESTAMP		Time at which the threshold was created.
ALTER_TIME	TIMESTAMP		Time at which the threshold was last altered.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.TRANSFORMS

Each row represents the functions that handle transformations between a user-defined type and a base SQL type, or the reverse.

Table 249. SYSCAT.TRANSFORMS Catalog View

Column Name	Data Type	Nullable	Description
TYPEID	SMALLINT		Identifier for the data type.
TYPESCHEMA	VARCHAR (128)		Schema name of the data type if TYPEMODULEID is null; otherwise schema name of the module to which the data type belongs.
TYPENAME	VARCHAR (128)		Unqualified name of the data type.
GROUPNAME	VARCHAR (128)		Name of the transform group.
FUNCID	INTEGER		Identifier for the routine.
FUNCSCHEMA	VARCHAR (128)		Schema name of the routine if ROUTINEMODULEID is null; otherwise schema name of the module to which the routine belongs.
FUNCNAME	VARCHAR (128)		Unqualified name of the routine.
SPECIFICNAME	VARCHAR (128)		Name of the routine instance (might be system-generated).
TRANSFORMTYPE	VARCHAR(8)		• 'FROM SQL' = Transform function transforms a structured type from SQL • 'TO SQL' = Transform function transforms a structured type to SQL
FORMAT	CHAR (1)		Format produced by the FROM SQL transform. • S = Structured data type • U = User-defined
MAXLENGTH	INTEGER	Y	Maximum length (in bytes) of output from the FROM SQL transform; the null value for TO SQL transforms.
ORIGIN	CHAR (1)		Source of this group of transforms. • O = Original transform group (built-in or system-defined) • R = Redefined transform group (only built-in groups can be redefined)
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.TRIGDEP

Each row represents a dependency of a trigger on some other object. The trigger depends on the object of type BTYPE of name BNAME, so a change to the object affects the trigger.

Table 250. SYSCAT.TRIGDEP Catalog View

Column Name	Data Type	Nullable	Description
TRIGSCHEMA	VARCHAR (128)		Schema name of the trigger.
TRIGNAME	VARCHAR (128)		Unqualified name of the trigger.

Table 250. SYSCAT.TRIGDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
BTYPE	CHAR (1)		Type of object on which there is a dependency. Possible values are: • A = Table alias • B = Trigger • F = Routine • G = Global temporary table • H = Hierachy table • K = Package • L = Detached table • N = Nickname • O = Privilege dependency on all subtables or subviews in a table or view hierarchy • Q = Sequence • R = User-defined data type • S = Materialized query table • T = Table (not typed) • U = Typed table • V = View (not typed) • W = Typed view • X = Index extension • Z = XSR object • q = Sequence alias • u = Module alias • v = Global variable • * = Anchored to the row of a base table
BSCHEMA	VARCHAR (128)		Schema name of the object on which there is a dependency.
BMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the object on which a dependency belongs. The null value if not a module object.
BNAME	VARCHAR (128)		Unqualified name of the object on which there is a dependency. For routines (BTYPE = 'F'), this is the specific name.
BMODULEID	INTEGER	Y	Identifier for the module of the object on which there is a dependency.
TABAUTH	SMALLINT	Y	If BTYPE = 'O', 'S', 'T', 'U', 'V', 'W', or 'v', encodes the privileges on the table or view that are required by a dependent trigger; null value otherwise.

SYSCAT.TRIGGERS

Each row represents a trigger. For table hierarchies, each trigger is recorded only at the level of the hierarchy where the trigger was created.

Table 251. SYSCAT.TRIGGERS Catalog View

Column Name	Data Type	Nullable	Description
TRIGSCHEMA	VARCHAR (128)		Schema name of the trigger.
TRIGNAME	VARCHAR (128)		Unqualified name of the trigger.
OWNER	VARCHAR (128)		Authorization ID of the owner of the trigger.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
TABSCHEMA	VARCHAR (128)		Schema name of the table or view to which this trigger applies.
TABNAME	VARCHAR (128)		Unqualified name of the table or view to which this trigger applies.
TRIGTIME	CHAR (1)		Time at which triggered actions are applied to the base table, relative to the event that fired the trigger. • A = Trigger is applied after the event • B = Trigger is applied before the event • I = Trigger is applied instead of the event
TRIGEVENT	CHAR (1)		Event that fires the trigger. • D = Delete operation • I = Insert operation • U = Update operation
GRANULARITY	CHAR (1)		Trigger is executed once per: • R = Row • S = Statement
VALID	CHAR (1)		• N = Trigger is invalid • X = Trigger is inoperative and must be recreated • Y = Trigger is valid
CREATE_TIME	TIMESTAMP		Time at which the trigger was defined. Used in resolving functions and types.
QUALIFIER	VARCHAR (128)		Value of the default schema at the time of object definition. Used to complete any unqualified references.
FUNC_PATH	CLOB (2K)		SQL path in effect when the trigger was defined.
TEXT	CLOB (2M)		Full text of the CREATE TRIGGER statement, exactly as typed.
LAST_REGEN_TIME	TIMESTAMP		Time at which the packed descriptor for the trigger was last regenerated.
COLLATIONSCHEMA	VARCHAR (128)		Schema name of the collation for the trigger.
COLLATIONNAME	VARCHAR (128)		Unqualified name of the collation for the trigger.
COLLATIONSCHEMA_ORDERBY	VARCHAR (128)		Schema name of the collation for ORDER BY clauses in the trigger.
COLLATIONNAME_ORDERBY	VARCHAR (128)		Unqualified name of the collation for ORDER BY clauses in the trigger.
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the trigger.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.TYPEMAPPINGS

Each row represents a data type mapping between a locally-defined data type and a data source data type. There are two mapping types (mapping directions):

- Forward type mappings map a data source data type to a locally-defined data type.
- Reverse type mappings map a locally-defined data type to a data source data type.

Table 252. SYSCAT.TYPEMAPPINGS Catalog View

Column Name	Data Type	Nullable	Description
TYPE_MAPPING	VARCHAR (18)		Name of the type mapping (might be system-generated).
MAPPINGDIRECTION	CHAR (1)		Indicates whether this type mapping is a forward or a reverse type mapping. • F = Forward type mapping • R = Reverse type mapping
TYPESCHEMA	VARCHAR (128)	Y	Schema name of the local type in a data type mapping; the null value for built-in types.
TYPENAME	VARCHAR (128)		Unqualified name of the local type in a data type mapping.
TYPEID	SMALLINT		Identifier for the data type.
SOURCETYPEID	SMALLINT		Identifier for the source type.
OWNER	VARCHAR (128)		Authorization ID of the owner of the type mapping. 'SYSIBM' indicates a built-in type mapping.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
LENGTH	INTEGER	Y	Maximum length or precision of the local data type in this mapping. If the null value, the system determines the maximum length or precision. For character types, represents the maximum number of bytes.
SCALE	SMALLINT	Y	Maximum number of digits in the fractional part of a local decimal value or the maximum number of digits of fractional seconds of a local TIMESTAMP value in this mapping. If the null value, the system determines the maximum number.
LOWER_LEN	INTEGER	Y	Minimum length or precision of the local data type in this mapping. If the null value, the system determines the minimum length or precision. For character types, represents the minimum number of bytes.
UPPER_LEN	INTEGER	Y	Maximum length or precision of the local data type in this mapping. If the null value, the system determines the maximum length or precision. For character types, represents the maximum number of bytes.

Table 252. SYSCAT.TYPEMAPPINGS Catalog View (continued)

Column Name	Data Type	Nullable	Description
LOWER_SCALE	SMALLINT	Y	Minimum number of digits in the fractional part of a local decimal value or the minimum number of digits of fractional seconds of a local TIMESTAMP value in this mapping. If the null value, the system determines the minimum number.
UPPER_SCALE	SMALLINT	Y	Maximum number of digits in the fractional part of a local decimal value or the maximum number of digits of fractional seconds of a local TIMESTAMP value in this mapping. If the null value, the system determines the maximum number.
S_OPR_P	CHAR (2)	Y	Relationship between the scale and precision of a local decimal value in this mapping. Basic comparison operators (=, <, >, <=, >=, <>) can be used. A null value indicates that no specific relationship is required.
BIT_DATA	CHAR (1)	Y	Indicates whether or not this character type is for bit data. Possible values are: • N = This type is not for bit data • Y = This type is for bit data • Null value = This is not a character data type, or the system determines the bit data attribute
WRAPNAME	VARCHAR (128)	Y	Data access protocol (wrapper) to which this mapping applies.
SERVERNAME	VARCHAR (128)	Y	Uppercase name of the server.
SERVERTYPE	VARCHAR (30)	Y	Type of server.
SERVERVERSION	VARCHAR (18)	Y	Server version.
REMOTE_TYPESCHEMA	VARCHAR (128)	Y	Schema name of the data source data type.
REMOTE_TYPENAME	VARCHAR (128)		Unqualified name of the data source data type.
REMOTE_META_TYPE	CHAR (1)	Y	Indicates whether this remote type is a system built-in type or a distinct type. • S = System built-in type • T = Distinct type
REMOTE_LOWER_LEN	INTEGER	Y	Minimum length or precision of the remote data type in this mapping, or the null value. For character types, represents the minimum number of characters (not bytes). For binary types, represents the minimum number of bytes. A value of -1 indicates that the default length or precision is used, or that the remote type does not have a length or precision.

Table 252. SYSCAT.TYPEMAPPINGS Catalog View (continued)

Column Name	Data Type	Nullable	Description
REMOTE_UPPER_LEN	INTEGER	Y	Maximum length or precision of the remote data type in this mapping, or the null value. For character types, represents the maximum number of characters (not bytes). For binary types, represents the maximum number of bytes. A value of -1 indicates that the default length or precision is used, or that the remote type does not have a length or precision.
REMOTE_LOWER_SCALE	SMALLINT	Y	Minimum number of digits in the fractional part of a remote decimal value or the minimum number of digits of fractional seconds of a remote TIMESTAMP value in this mapping, or the null value.
REMOTE_UPPER_SCALE	SMALLINT	Y	Maximum number of digits in the fractional part of a remote decimal value or the maximum number of digits of fractional seconds of a remote TIMESTAMP value in this mapping, or the null value.
REMOTE_S_OPR_P	CHAR (2)	Y	Relationship between the scale and precision of a remote decimal value in this mapping. Basic comparison operators (=, <, >, <=, >=, <>) can be used. A null value indicates that no specific relationship is required.
REMOTE_BIT_DATA	CHAR (1)	Y	Indicates whether or not this remote character type is for bit data. Possible values are: • N = This type is not for bit data • Y = This type is for bit data • Null value = This is not a character data type, or the system determines the bit data attribute
USER_DEFINED	CHAR (1)		Indicates whether or not the mapping is user-defined. The value is always 'Y'; that is, the mapping is always user-defined.
CREATE_TIME	TIMESTAMP		Time at which this mapping was created.
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the type mapping. 'SYSIBM' indicates a built-in type mapping.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.VARIABLEAUTH

Each row represents a user, group, or role that has been granted one or more privileges by a specific grantor on a global variable in the database that is not defined in a module.

Table 253. SYSCAT.VARIABLEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the privilege.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
VARSCHEMA	VARCHAR (128)		Schema name of the global variable if VARMODULEID is null; otherwise schema name of the module to which the global variable belongs.
VARNAME	VARCHAR (128)		Unqualified name of the global variable.
VARID	INTEGER		Identifier for the global variable.
READAUTH	CHAR (1)		Privilege to read the global variable. • G = Held and grantable • N = Not held • Y = Held
WRITEAUTH	CHAR (1)		Privilege to write the global variable. • G = Held and grantable • N = Not held • Y = Held

SYSCAT.VARIABLEDEP

Each row represents a dependency of a global variable on some other object. The global variable depends on the object of type BTYPE of name BNAME, so a change to the object affects the global variable.

Table 254. SYSCAT.VARIABLEDEP Catalog View

Column Name	Data Type	Nullable	Description
VARSCHEMA	VARCHAR (128)		Schema name of the global variable that has dependencies on another object.
VARMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the global variable belongs. The null value if not a module variable.
VARNAME	VARCHAR (128)		Unqualified name of the global variable that has dependencies on another object.
VARMODULEID	INTEGER	Y	Identifier for the module of the object that has dependencies on another object.

Table 254. SYSCAT.VARIABLEDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
BTYPE	CHAR (1)		Type of object on which there is a dependency. Possible values are: • A = Table alias • F = Routine • G = Global temporary table • H = Hierarchy table • N = Nickname • O = Privilege dependency on all subtables or subviews in a table or view hierarchy • R = User-defined data type • S = Materialized query table • T = Table (not typed) • U = Typed table • V = View (not typed) • W = Typed view • q = Sequence alias • u = Module alias • v = Global variable • * = Anchored to the row of a base table
BSCHEMA	VARCHAR (128)		Schema name of the object on which there is a dependency.
BMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the object on which a dependency belongs. The null value if not a module object.
BNAME	VARCHAR (128)		Unqualified name of the object on which there is a dependency. For routines (BTYPE = 'F'), this is the specific name.
BMODULEID	INTEGER	Y	Identifier for the module of the object on which there is a dependency.
TABAUTH	SMALLINT	Y	If BTYPE = 'O', 'S', 'T', 'U', 'V', 'W', or 'v', encodes the privileges on the table or view that are required by the dependent global variable; the null value otherwise.

SYSCAT.VARIABLES

Each row represents a global variable.

Table 255. SYSCAT.VARIABLES Catalog View

Column Name	Data Type	Nullable	Description
VARSCHEMA	VARCHAR (128)		Schema name of the global variable if VARMODULEID is null; otherwise schema name of the module to which the global variable belongs.
VARMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the global variable belongs. The null value if not a module variable.
VARNAME	VARCHAR (128)		Unqualified name of the global variable.

Table 255. SYSCAT.VARIABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
VARMODULEID	INTEGER	Y	Identifier for the module to which the global variable belongs. The null value if not a module variable.
VARID	INTEGER		Identifier for the global variable.
OWNER	VARCHAR (128)		Authorization ID of the owner of the global variable.
OWNERTYPE	CHAR (1)		• U = The owner is an individual user
CREATE_TIME	TIMESTAMP		Time at which the global variable was created.
LAST_REGEN_TIME	TIMESTAMP		Time at which the default expression was last regenerated.
VALID	CHAR (1)		• N = The global variable is invalid • Y = The global variable is valid
PUBLISHED	CHAR (1)		Indicates whether the module variable can be referenced outside its module. • N = The module variable is not published • Y = The module variable is published • Blank = Not applicable
TYPESCHEMA	VARCHAR (128)		Schema name of the data type if TYPEMODULEID is null; otherwise schema name of the module to which the data type belongs.
TYPEMODULENAME	VARCHAR (128)		Unqualified name of the module to which the variable data type belongs. The null value if the variable data type does not belong to a module.
TYPENAME	VARCHAR (128)		Unqualified name of the data type.
TYPEMODULEID	INTEGER	Y	Identifier for the module to which the variable data type belongs. The null value if the variable data type does not belong to a module.
LENGTH	INTEGER		Maximum length of the global variable.
SCALE	SMALLINT		Scale if the global variable data type is DECIMAL or distinct type based on DECIMAL; the number of digits of fractional seconds if the global variable data type is TIMESTAMP or distinct type based on TIMESTAMP; 0 otherwise.
CODEPAGE	SMALLINT		Code page of the global variable.
COLLATIONSCHEMA	VARCHAR (128)		Schema name of the collation for the variable.
COLLATIONNAME	VARCHAR (128)		Unqualified name of the collation for the variable.
COLLATIONSCHEMA_ORDERBY	VARCHAR (128)		Schema name of the collation for ORDER BY clauses in the variable.
COLLATIONNAME_ORDERBY	VARCHAR (128)		Unqualified name of the collation for ORDER BY clauses in the variable.
SCOPE	CHAR (1)		Scope of the global variable. • S = Session
DEFAULT	CLOB (64K)	Y	Expression used to calculate the initial value of the global variable when first referenced.
QUALIFIER	VARCHAR (128)	Y	Value of the default schema at the time the variable was defined.
FUNC_PATH	CLOB (2K)	Y	SQL path in effect when the variable was defined.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.
READONLY	CHAR (1)		• C = Read-only because the global variable is defined with a CONSTANT clause • N = Not read-only

SYSCAT.VIEWS

Each row represents a view or materialized query table.

Table 256. SYSCAT.VIEWS Catalog View

Column Name	Data Type	Nullable	Description
VIEWSCHEMA	VARCHAR (128)		Schema name of the view or materialized query table.
VIEWNAME	VARCHAR (128)		Unqualified name of the view or materialized query table.
OWNER	VARCHAR (128)		Authorization ID of the owner of the view or materialized query table.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
SEQNO	SMALLINT		Always 1.
VIEWCHECK	CHAR (1)		Type of view checking. • C = Cascaded check option • L = Local check option • N = No check option or is a materialized query table
READONLY	CHAR (1)		• N = View can be updated by users with appropriate authorization or is a materialized query table • Y = View is read-only because of its definition
VALID	CHAR (1)		• N = View or materialized query table definition is invalid • X = View or materialized query table definition is inoperative and must be recreated • Y = View or materialized query table definition is valid
QUALIFIER	VARCHAR (128)		Value of the default schema at the time of object definition. Used to complete any unqualified references.
FUNC_PATH	CLOB (2K)		SQL path in effect when the view or materialized query table was defined.
TEXT	CLOB (2M)		Full text of the view or materialized query table CREATE statement, exactly as typed.
DEFINER[1]	VARCHAR (128)		Authorization ID of the owner of the view or materialized query table.

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.WORKACTIONS

Each row represents a work action that is defined for a work action set.

Table 257. SYSCAT.WORKACTIONS Catalog View

Column Name	Data Type	Nullable	Description
ACTIONNAME	VARCHAR (128)		Name of the work action.
ACTIONID	INTEGER		Identifier for the work action.
ACTIONSETNAME	VARCHAR (128)	Y	Name of the work action set.
ACTIONSETID	INTEGER		Identifier of the work action set to which this work action belongs. This column refers to the ACTIONSETID column in the SYSCAT.WORKACTIONSETS view.
WORKCLASSNAME	VARCHAR (128)	Y	Name of the work class.
WORKCLASSID	INTEGER		Identifier of the work class. This column refers to the WORKCLASSID column in the SYSCAT.WORKCLASSES view.
CREATE_TIME	TIMESTAMP		Time at which the work action was created.
ALTER_TIME	TIMESTAMP		Time at which the work action was last altered.
ENABLED	CHAR (1)		• N = This work action is disabled. • Y = This work action is enabled.

Table 257. SYSCAT.WORKACTIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
ACTIONTYPE	CHAR (1)		The type of action performed on each DB2 activity that matches the attributes in the work class within scope. • B = Collect basic aggregate activity data, specifiable only for work action sets that apply to service classes. • C = Allow any DB2 activity under the associated work class to execute and increment the work class counter. • D = Collect activity data with details at the database partition of the coordinator of the activity. • E = Collect extended aggregate activity data, specifiable only for work action sets that apply to service classes. • F = Collect activity data with details, section, and values at the database partition of the coordinator of the activity. • G = Collect activity details and section at the database partition of the coordinator of the activity and collect activity data at all database partitions. • H = Collect activity details, section, and values at the database partition of the coordinator of the activity and collect activity data at all database partitions. • M = Map to a service subclass, specifiable only for work action sets that apply to service classes. • P = Prevent the execution of any DB2 activity under the work class with which this work action is associated. • S = Collect activity data with details and section at the database partition of the coordinator of the activity. • T = The action represents a threshold, specifiable only for work action sets that are associated with a database. • U = Map all activities with a nesting level of zero and all activities nested under these activities to a service subclass, specifiable only for work action sets that apply to service classes. • V = Collect activity data with details and values at the coordinator partition. • W = Collect activity data without details at the coordinator partition. • X = Collect activity data with details at the coordinator partition and collect activity data at all database partitions. • Y = Collect activity data with details and values at the coordinator partition and collect activity data at all database partitions. • Z = Collect activity data without details at all database partitions.

Table 257. SYSCAT.WORKACTIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
REFOBJECTID	INTEGER	Y	If ACTIONTYPE is 'M' (map) or 'N' (map nested), this value is set to the ID of the service subclass to which the DB2 activity is mapped. If ACTIONTYPE is 'T' (threshold), this value is set to the ID of the threshold to be used. For all other actions, this value is NULL.
REFOBJECTTYPE	VARCHAR (30)		If the ACTIONTYPE is 'M' or 'N', this value is set to 'SERVICE CLASS'; if the ACTIONTYPE is 'T', this value is 'THRESHOLD'; the null value otherwise.

SYSCAT.WORKACTIONSETS

Each row represents a work action set.

Table 258. SYSCAT.WORKACTIONSETS Catalog View

Column Name	Data Type	Nullable	Description
ACTIONSETNAME	VARCHAR (128)		Name of the work action set.
ACTIONSETID	INTEGER		Identifier for the work action set.
WORKCLASSSETNAME	VARCHAR (128)	Y	Name of the work class set.
WORKCLASSSETID	INTEGER		The identifier of the work class set that is to be mapped to the object specified by the OBJECTID. This column refers to WORKCLASSSETID in the SYSCAT.WORKCLASSSETS view.
CREATE_TIME	TIMESTAMP		Time at which the work action set was created.
ALTER_TIME	TIMESTAMP		Time at which the work action set was last altered.
ENABLED	CHAR (1)		• N = This work action set is disabled. • Y = This work action set is enabled.
OBJECTTYPE	CHAR (1)		• b = Service superclass • Blank = Database
OBJECTNAME	VARCHAR (128)	Y	Name of the service class.
OBJECTID	INTEGER		The identifier of the object to which the work class set (specified by the WORKCLASSSETID) is mapped. If the OBJECTTYPE is blank, the OBJECTID is -1. If the OBJECTTYPE is 'b', the OBJECTID is the ID of the service superclass.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.WORKCLASSES

Each row represents a work class defined for a work class set.

Table 259. SYSCAT.WORKCLASSES Catalog View

Column Name	Data Type	Nullable	Description
WORKCLASSNAME	VARCHAR (128)		Name of the work class.
WORKCLASSSETNAME	VARCHAR (128)	Y	Name of the work class set.
WORKCLASSID	INTEGER		Identifier for the work class.
WORKCLASSSETID	INTEGER		Identifier for the work class set to which this work class belongs. This column refers to the WORKCLASSSETID column in the SYSCAT.WORKCLASSSETS view.
CREATE_TIME	TIMESTAMP		Time at which the work class was created.
ALTER_TIME	TIMESTAMP		Time at which the work class was last altered.
WORKTYPE	SMALLINT		The type of DB2 activity. • 1 = ALL • 2 = READ • 3 = WRITE • 4 = CALL • 5 = DML • 6 = DDL • 7 = LOAD
RANGEUNITS	CHAR (1)		The units to use for the bottom and top range. • C = Cardinality • T = Timerons • Blank = Not applicable
FROMVALUE	DOUBLE	Y	The low value of the range in the units specified by the RANGEUNITS. Null value when RANGEUNITS is blank.
TOVALUE	DOUBLE	Y	The high value of the range in the units specified by the RANGEUNITS. Null value when RANGEUNITS is blank. -1 value is used to indicate no upper bound.
ROUTINESCHEMA	VARCHAR (128)	Y	Schema name of the procedures that are called from the CALL statement. Null value when WORKTYPE is not 4 (CALL) or 1 (ALL).
INITIALSQLDATAPRIORITY	CHAR (1)		Reserved for future use.
EVALUATIONORDER	SMALLINT		Uniquely identifies the evaluation order used for choosing a work class within a work class set.

SYSCAT.WORKCLASSSETS

Each row represents a work class set.

Table 260. SYSCAT.WORKCLASSSETS Catalog View

Column Name	Data Type	Nullable	Description
WORKCLASSSETNAME	VARCHAR (128)		Name of the work class set.
WORKCLASSSETID	INTEGER		Identifier for the work class set.

Table 260. SYSCAT.WORKCLASSSETS Catalog View (continued)

Column Name	Data Type	Nullable	Description
CREATE_TIME	TIMESTAMP		Time at which the work class set was created.
ALTER_TIME	TIMESTAMP		Time at which the work class set was last altered.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.WORKLOADAUTH

Each row represents a user, group, or role that has been granted USAGE privilege on a workload.

Table 261. SYSCAT.WORKLOADAUTH Catalog View

Column Name	Data Type	Nullable	Description
WORKLOADID	INTEGER		Identifier for the workload.
WORKLOADNAME	VARCHAR (128)		Name of the workload.
GRANTOR	VARCHAR (128)		Grantor of the privilege.
GRANTORTYPE	CHAR (1)		• U = Grantee is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
USAGEAUTH	CHAR (1)		Indicates whether grantee holds USAGE privilege on the workload. • N = Not held • Y = Held

SYSCAT.WORKLOADCONNATTR

Each row represents a connection attribute in the definition of a workload.

Table 262. SYSCAT.WORKLOADCONNATTR Catalog View

Column Name	Data Type	Nullable	Description
WORKLOADID	INTEGER		Identifier for the workload.
WORKLOADNAME	VARCHAR (128)		Name of the workload.

Table 262. SYSCAT.WORKLOADCONNATTR Catalog View (continued)

Column Name	Data Type	Nullable	Description
CONNATTRTYPE	VARCHAR (30)		Type of the connection attribute. • 1 = APPLNAME • 2 = SYSTEM_USER • 3 = SESSION_USER • 4 = SESSION_USER GROUP • 5 = SESSION_USER ROLE • 6 = CURRENT CLIENT_USERID • 7 = CURRENT CLIENT_APPLNAME • 8 = CURRENT CLIENT_WRKSTNNAME • 9 = CURRENT CLIENT_ACCTNG • 10 = ADDRESS
CONNATTRVALUE	VARCHAR (1000)		Value of the connection attribute.

SYSCAT.WORKLOADS

Each row represents a workload.

Table 263. SYSCAT.WORKLOADS Catalog View

Column Name	Data Type	Nullable	Description
WORKLOADID	INTEGER		Identifier for the workload.
WORKLOADNAME	VARCHAR (128)		Name of the workload.
EVALUATIONORDER	SMALLINT		Evaluation order used for choosing a workload.
CREATE_TIME	TIMESTAMP		Time at which the workload was created.
ALTER_TIME	TIMESTAMP		Time at which the workload was last altered.
ENABLED	CHAR (1)		• N = This workload is disabled. • Y = This workload is enabled.
ALLOWACCESS	CHAR (1)		• N = A UOW associated with this workload will be rejected. • Y = A unit of work (UOW) associated with this workload can access the database.
SERVICECLASSNAME	VARCHAR (128)		Name of the service subclass to which a unit of work (associated with this workload) is assigned.
PARENTSERVICECLASSNAME	VARCHAR (128)	Y	Name of the service superclass to which a unit of work (associated with this workload) is assigned.
COLLECTAGGACTDATA	CHAR (1)		Specifies what aggregate activity data should be captured for the workload by the applicable event monitor. • B = Collect base aggregate activity data • E = Collect extended aggregate activity data • N = None

Table 263. SYSCAT.WORKLOADS Catalog View (continued)

Column Name	Data Type	Nullable	Description
COLLECTACTDATA	CHAR (1)		Specifies what activity data should be collected by the applicable event monitor. • D = Activity data with details • N = None • S = Activity data with details and section environment • V = Activity data with details and values. Applies when the COLLECT column is set to 'C' • W = Activity data without details • X = Activity data with details, section environment, and values
COLLECTACTPARTITION	CHAR (1)		Specifies where activity data is collected. • C = Database partition of the coordinator of the activity • D = All database partitions
COLLECTDEADLOCK	CHAR (1)		Specifies that deadlock events should be collect by the applicable event monitor. • H = Collect deadlock data with past activities only • N = Do not not collect deadlock data • V = Collect deadlock data with past activities and values • W = Collect deadlock data without past activities and values
COLLECTLOCKTIMEOUT	CHAR (1)		Specifies that lock timeout events should be collect by the applicable event monitor. • H = Collect lock timeout data with past activities only • N = Do not not collect lock timeout data • V = Collect lock timeout data with past activities and values • W = Collect lock timeout data without past activities and values
COLLECTLOCKWAIT	CHAR (1)		Specifies that lock wait events should be collect by the applicable event monitor. • H = Collect lock wait data with past activities only • N = Do not not collect lock wait data • V = Collect lock wait data with past activities and values • W = Collect lock wait data without past activities and values
LOCKWAITVALUE	INTEGER		Specifies the time in milliseconds a lock should wait before a lock event is collected by the applicable event monitor; 0 when COLLECTLOCKWAIT = 'N'

Table 263. SYSCAT.WORKLOADS Catalog View (continued)

Column Name	Data Type	Nullable	Description
COLLECTACTMETRICS	CHAR (1)		Specifies the monitoring level for activities submitted by an occurrence of the workload. • B = Collect base activity metrics • E = Collect extended activity metrics • N = None
COLLECTUOWDATA	CHAR (1)		Specifies what unit of work data should be collected by the applicable event monitor. • B = Collect base unit of work data • N = None
EXTERNALNAME	VARCHAR (128)	Y	Reserved for future use.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.WRAPOPTIONS

Each row represents a wrapper-specific option.

Table 264. SYSCAT.WRAPOPTIONS Catalog View

Column Name	Data Type	Nullable	Description
WRAPNAME	VARCHAR (128)		Name of the wrapper.
OPTION	VARCHAR (128)		Name of the wrapper option.
SETTING	VARCHAR (2048)		Value of the wrapper option.

SYSCAT.WRAPPERS

Each row represents a registered wrapper.

Table 265. SYSCAT.WRAPPERS Catalog View

Column Name	Data Type	Nullable	Description
WRAPNAME	VARCHAR (128)		Name of the wrapper.
WRAPTYPE	CHAR (1)		Type of wrapper. • N = Non-relational • R = Relational
WRAPVERSION	INTEGER		Version of the wrapper.
LIBRARY	VARCHAR (255)		Name of the file that contains the code used to communicate with the data sources that are associated with this wrapper.
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSCAT.XDBMAPGRAPHS

Each row represents a schema graph for an XDB map (XSR object).

Table 266. SYSCAT.XDBMAPGRAPHS Catalog View

Column Name	Data Type	Nullable	Description
OBJECTID	BIGINT		Unique generated identifier for an XSR object.
OBJECTSCHEMA	VARCHAR (128)		Schema name of the XSR object.
OBJECTNAME	VARCHAR (128)		Unqualified name of the XSR object.
SCHEMAGRAPHID	INTEGER		Schema graph identifier, which is unique within an XDB map identifier.
NAMESPACE	VARCHAR (1001)	Y	String identifier for the namespace URI of the root element.
ROOTELEMENT	VARCHAR (1001)	Y	String identifier for the element name of the root element.

SYSCAT.XDBMAPSHREDTREES

Each row represents one shred tree for a given schema graph identifier.

Table 267. SYSCAT.XDBMAPSHREDTREES Catalog View

Column Name	Data Type	Nullable	Description
OBJECTID	BIGINT		Unique generated identifier for an XSR object.
OBJECTSCHEMA	VARCHAR (128)		Schema name of the XSR object.
OBJECTNAME	VARCHAR (128)		Unqualified name of the XSR object.
SCHEMAGRAPHID	INTEGER		Schema graph identifier, which is unique within an XDB map identifier.
SHREDTREEID	INTEGER		Shred tree identifier, which is unique within an XDB map identifier.
MAPPINGDESCRIPTION	CLOB (1M)	Y	Diagnostic mapping information.

SYSCAT.XMLSTRINGS

Each row represents a single string and its unique string ID, used to condense structural XML data. The string is provided in both UTF-8 encoding and database codepage encoding.

Table 268. SYSCAT.XMLSTRINGS Catalog View

Column Name	Data Type	Nullable	Description
STRINGID	INTEGER		Unique string ID.
STRING	VARCHAR(1001)		The string represented in the database codepage.
STRING_UTF8	VARCHAR(1001)		The string in UTF-8 encoding (as stored in the catalog table).

SYSCAT.XSROBJECTAUTH

Each row represents a user, group, or role that has been granted the USAGE privilege on a particular XSR object.

Table 269. SYSCAT.XSROBJECTAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR (128)		Grantor of the privilege.
GRANTORTYPE	CHAR (1)		• S = Grantor is the system • U = Grantor is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
OBJECTID	BIGINT		Identifier for the XSR object.
USAGEAUTH	CHAR (1)		Privilege to use the XSR object and its components. • N = Not held • Y = Held

SYSCAT.XSROBJECTCOMPONENTS

Each row represents an XSR object component.

Table 270. SYSCAT.XSROBJECTCOMPONENTS Catalog View

Column Name	Data Type	Nullable	Description
OBJECTID	BIGINT		Unique generated identifier for an XSR object.
OBJECTSCHEMA	VARCHAR (128)		Schema name of the XSR object.
OBJECTNAME	VARCHAR (128)		Unqualified name of the XSR object.
COMPONENTID	BIGINT		Unique generated identifier for an XSR object component.
TARGETNAMESPACE	VARCHAR (1001)	Y	String identifier for the target namespace.
SCHEMALOCATION	VARCHAR (1001)	Y	String identifier for the schema location.
COMPONENT	BLOB (30M)		External representation of the component.
CREATE_TIME	TIMESTAMP		Time at which the XSR object component was registered.
STATUS	CHAR (1)		Registration status. • C = Complete • I = Incomplete

SYSCAT.XSROBJECTDETAILS

Each row represents an XML schema repository object.

Table 271. SYSCAT.XSROBJECTDETAILS Catalog View

Column Name	Data Type	Nullable	Description
OBJECTID	BIGINT		Unique generated identifier for an XML schema object.
OBJECTSCHEMA	VARCHAR (128)		Schema name of the XML schema object.
OBJECTNAME	VARCHAR (128)		Unqualified name of the XML schema object.

Table 271. SYSCAT.XSROBJECTDETAILS Catalog View (continued)

Column Name	Data Type	Nullable	Description
GRAMMAR	BLOB (127M)	Y	Binary representation of the grammar for the XML schema object.
PROPERTIES	BLOB (4190000)	Y	Properties document for the XML schema object.

SYSCAT.XSROBJECTDEP

Each row represents a dependency of an XSR object on some other object. The XSR object depends on the object of type BTYPE of name BNAME, so a change to the object affects the XSR object.

Table 272. SYSCAT.XSROBJECTDEP Catalog View

Column Name	Data Type	Nullable	Description
OBJECTID	BIGINT		Unique generated identifier for an XSR object.
OBJECTSCHEMA	VARCHAR (128)		Schema name of the XSR object.
OBJECTNAME	VARCHAR (128)		Unqualified name of the XSR object.
BTYPE	CHAR (1)		Type of object on which there is a dependency. Possible values are: • A = Table alias • B = Trigger • F = Routine • G = Global temporary table • H = Hierachy table • K = Package • L = Detached table • N = Nickname • O = Privilege dependency on all subtables or subviews in a table or view hierarchy • Q = Sequence • R = User-defined data type • S = Materialized query table • T = Table (not typed) • U = Typed table • V = View (not typed) • W = Typed view • X = Index extension • Z = XSR object • q = Sequence alias • u = Module alias • v = Global variable • * = Anchored to the row of a base table
BSCHEMA	VARCHAR (128)		Schema name of the object on which there is a dependency.

Table 272. SYSCAT.XSROBJECTDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
BMODULENAME	VARCHAR(128)	Y	Unqualified name of the module to which the object on which a dependency belongs. The null value if not a module object.
BNAME	VARCHAR (128)		Unqualified name of the object on which there is a dependency. For routines (BTYPE = 'F'), this is the specific name.
BMODULEID	INTEGER	Y	Identifier for the module of the object on which there is a dependency.
TABAUTH	SMALLINT	Y	If BTYPE = 'O', 'S', 'T', 'U', 'V', 'W', or 'v', encodes the privileges on the table or view that are required by a dependent trigger; null value otherwise.

SYSCAT.XSROBJECTHIERARCHIES

Each row represents the hierarchical relationship between an XSR object and its components.

Table 273. SYSCAT.XSROBJECTHIERARCHIES Catalog View

Column Name	Data Type	Nullable	Description
OBJECTID	BIGINT		Identifier for an XSR object.
COMPONENTID	BIGINT		Identifier for an XSR component.
HTYPE	CHAR (1)		Hierarchy type. • D = Document • N = Top-level namespace • P = Primary document
TARGETNAMESPACE	VARCHAR (1001)	Y	String identifier for the component's target namespace.
SCHEMALOCATION	VARCHAR (1001)	Y	String identifier for the component's schema location.

SYSCAT.XSROBJECTS

Each row represents an XML schema repository object.

Table 274. SYSCAT.XSROBJECTS Catalog View

Column Name	Data Type	Nullable	Description
OBJECTID	BIGINT		Unique generated identifier for an XSR object.
OBJECTSCHEMA	VARCHAR (128)		Schema name of the XSR object.
OBJECTNAME	VARCHAR (128)		Unqualified name of the XSR object.
TARGETNAMESPACE	VARCHAR (1001)	Y	String identifier for the target namespace, or public identifier.
SCHEMALOCATION	VARCHAR (1001)	Y	String identifier for the schema location, or system identifier.
OBJECTINFO	XML	Y	Metadata document.

Table 274. SYSCAT.XSROBJECTS Catalog View (continued)

Column Name	Data Type	Nullable	Description
OBJECTTYPE	CHAR (1)		XSR object type. • D = DTD • E = External entity • S = XML schema
OWNER	VARCHAR (128)		Authorization ID of the owner of the XSR object.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
CREATE_TIME	TIMESTAMP		Time at which the object was registered.
ALTER_TIME	TIMESTAMP		Time at which the object was last updated (replaced).
STATUS	CHAR (1)		Registration status. • C = Complete • I = Incomplete • R = Replace • T = Temporary
DECOMPOSITION	CHAR (1)		Indicates whether or not decomposition (shredding) is enabled on this XSR object. • N = Not enabled • X = Inoperative • Y = Enabled
REMARKS	VARCHAR (254)	Y	User-provided comments, or the null value.

SYSIBM.SYSDUMMY1

Contains one row. This view is available for applications that require compatibility with DB2 for z/OS.

Table 275. SYSIBM.SYSDUMMY1 Catalog View

Column Name	Data Type	Nullable	Description
IBMREQD	CHAR(1)		'Y'

SYSSTAT.COLDIST

Each row represents the *n*th most frequent value of some column, or the *n*th quantile (cumulative distribution) value of the column. Applies to columns of real tables only (not views). No statistics are recorded for inherited columns of typed tables.

Table 276. SYSSTAT.COLDIST Catalog View

Column Name	Data Type	Nullable	Updat-able	Description
TABSCHEMA	VARCHAR (128)			Schema name of the table to which the statistics apply.
TABNAME	VARCHAR (128)			Unqualified name of the table to which the statistics apply.

Table 276. SYSSTAT.COLDIST Catalog View (continued)

Column Name	Data Type	Nullable	Updat-able	Description
COLNAME	VARCHAR (128)			Name of the column to which the statistics apply.
TYPE	CHAR (1)			• F = Frequency value • Q = Quantile value
SEQNO	SMALLINT			If TYPE = 'F', *n* in this column identifies the *n*th most frequent value. If TYPE = 'Q', *n* in this column identifies the *n*th quantile value.
COLVALUE[1]	VARCHAR (254)	Y	Y	Data value as a character literal or a null value.
VALCOUNT	BIGINT		Y	If TYPE = 'F', VALCOUNT is the number of occurrences of COLVALUE in the column. If TYPE = 'Q', VALCOUNT is the number of rows whose value is less than or equal to COLVALUE.
DISTCOUNT[2]	BIGINT	Y	Y	If TYPE = 'Q', this column records the number of distinct values that are less than or equal to COLVALUE (the null value if unavailable).

Note:

1. In the catalog view, the value of COLVALUE is always shown in the database code page and can contain substitution characters. However, the statistics are gathered internally in the code page of the column's table, and will therefore use actual column values when applied during query optimization.
2. DISTCOUNT is collected only for columns that are the first key column in an index.

SYSSTAT.COLGROUPDIST

Each row represents the value of the column in a column group that makes up the *n*th most frequent value of the column group or the *n*th quantile value of the column group.

Table 277. SYSSTAT.COLGROUPDIST Catalog View

Column Name	Data Type	Nullable	Updat-able	Description
COLGROUPID	INTEGER			Identifier for the column group.
TYPE	CHAR (1)			• F = Frequency value • Q = Quantile value
ORDINAL	SMALLINT			Ordinal number of the column in the column group.
SEQNO	SMALLINT			If TYPE = 'F', *n* in this column identifies the *n*th most frequent value. If TYPE = 'Q', *n* in this column identifies the *n*th quantile value.
COLVALUE	VARCHAR (254)		Y	Data value as a character literal or a null value.

SYSSTAT.COLGROUPDISTCOUNTS

Each row represents the distribution statistics that apply to the *n*th most frequent value of a column group or the *n*th quantile of a column group.

Table 278. SYSSTAT.COLGROUPDISTCOUNTS Catalog View

Column Name	Data Type	Nullable	Updat-able	Description
COLGROUPID	INTEGER			Identifier for the column group.
TYPE	CHAR (1)			• F = Frequency value • Q = Quantile value
SEQNO	SMALLINT			Sequence number *n* representing the *n*th TYPE value.
VALCOUNT	BIGINT		Y	If TYPE = 'F', VALCOUNT is the number of occurrences of COLVALUE for the column group with this SEQNO. If TYPE = 'Q', VALCOUNT is the number of rows whose value is less than or equal to COLVALUE for the column group with this SEQNO.
DISTCOUNT	BIGINT		Y	If TYPE = 'Q', this column records the number of distinct values that are less than or equal to COLVALUE for the column group with this SEQNO (the null value if unavailable).

SYSSTAT.COLGROUPS

Each row represents a column group and statistics that apply to the entire column group.

Table 279. SYSSTAT.COLGROUPS Catalog View

Column Name	Data Type	Nullable	Updat-able	Description
COLGROUPSCHEMA	VARCHAR (128)			Schema name of the column group.
COLGROUPNAME	VARCHAR (128)			Unqualified name of the column group.
COLGROUPID	INTEGER			Identifier for the column group.
COLGROUPCARD	BIGINT		Y	Cardinality of the column group.
NUMFREQ_VALUES	SMALLINT			Number of frequent values collected for the column group.
NUMQUANTILES	SMALLINT			Number of quantiles collected for the column group.

SYSSTAT.COLUMNS

Each row represents a column defined for a table, view, or nickname.

Table 280. SYSSTAT.COLUMNS Catalog View

Column Name	Data Type	Nullable	Updat-able	Description
TABSCHEMA	VARCHAR (128)			Schema name of the table, view, or nickname that contains the column.
TABNAME	VARCHAR (128)			Unqualified name of the table, view, or nickname that contains the column.
COLNAME	VARCHAR (128)			Name of the column.
COLCARD	BIGINT		Y	Number of distinct values in the column; -1 if statistics are not collected; -2 for inherited columns and columns of hierarchy tables.
HIGH2KEY[1]	VARCHAR (254)	Y	Y	Second-highest data value. Representation of numeric data changed to character literals. Empty if statistics are not collected. Empty for inherited columns and columns of hierarchy tables.
LOW2KEY[1]	VARCHAR (254)	Y	Y	Second-lowest data value. Representation of numeric data changed to character literals. Empty if statistics are not collected. Empty for inherited columns and columns of hierarchy tables.
AVGCOLLEN	INTEGER		Y	Average space in bytes when the column is stored in database memory or a temporary table. For LOB data types that are not inlined, LONG data types, and XML documents, the value used to calculate the average column length is the length of the data descriptor. An extra byte is required if the column is nullable; -1 if statistics have not been collected; -2 for inherited columns and columns of hierarchy tables. Note: The average space required to store the column on disk may be different than the value represented by this statistic.
NUMNULLS	BIGINT		Y	Number of null values in the column; -1 if statistics are not collected.
PCTINLINED	SMALLINT			Percentage of inlined XML documents or LOB data. -1 if statistics have not been collected.
SUB_COUNT	SMALLINT		Y	Average number of sub-elements in the column. Applicable to character string columns only.
SUB_DELIM_LENGTH	SMALLINT		Y	Average length of the delimiters that separate each sub-element in the column. Applicable to character string columns only.

Table 280. SYSSTAT.COLUMNS Catalog View (continued)

Column Name	Data Type	Nullable	Updat-able	Description
AVGCOLLENCHAR	INTEGER		Y	Average number of characters (based on the collation in effect for the column) required for the column; -1 if the data type of the column is long, LOB, or XML or if statistics have not been collected; -2 for inherited columns and columns of hierarchy tables.

Note:

1. In the catalog view, the values of HIGH2KEY and LOW2KEY are always shown in the database code page and can contain substitution characters. However, the statistics are gathered internally in the code page of the column's table, and will therefore use actual column values when applied during query optimization.

SYSSTAT.INDEXES

Each row represents an index. Indexes on typed tables are represented by two rows: one for the "logical index" on the typed table, and one for the "H-index" on the hierarchy table.

Table 281. SYSSTAT.INDEXES Catalog View

Column Name	Data Type	Nullable	Updat-able	Description
INDSCHEMA	VARCHAR (128)			Schema name of the index.
INDNAME	VARCHAR (128)			Unqualified name of the index.
TABSCHEMA	VARCHAR (128)			Schema name of the table or nickname on which the index is defined.
TABNAME	VARCHAR (128)			Unqualified name of the table or nickname on which the index is defined.
COLNAMES	VARCHAR (640)			This column is no longer used and will be removed in the next release.
NLEAF	BIGINT		Y	Number of leaf pages; -1 if statistics are not collected.
NLEVELS	SMALLINT		Y	Number of index levels; -1 if statistics are not collected.
FIRSTKEYCARD	BIGINT		Y	Number of distinct first-key values; -1 if statistics are not collected.
FIRST2KEYCARD	BIGINT		Y	Number of distinct keys using the first two columns of the index; -1 if statistics are not collected, or if not applicable.
FIRST3KEYCARD	BIGINT		Y	Number of distinct keys using the first three columns of the index; -1 if statistics are not collected, or if not applicable.
FIRST4KEYCARD	BIGINT		Y	Number of distinct keys using the first four columns of the index; -1 if statistics are not collected, or if not applicable.
FULLKEYCARD	BIGINT		Y	Number of distinct full-key values; -1 if statistics are not collected.

Table 281. SYSSTAT.INDEXES Catalog View (continued)

Column Name	Data Type	Nullable	Updat-able	Description
CLUSTERRATIO[4]	SMALLINT		Y	Degree of data clustering with the index; -1 if statistics are not collected or if detailed index statistics are collected (in which case, CLUSTERFACTOR will be used instead).
CLUSTERFACTOR[4]	DOUBLE		Y	Finer measurement of the degree of clustering; -1 if statistics are not collected or if the index is defined on a nickname.
SEQUENTIAL_PAGES	BIGINT		Y	Number of leaf pages located on disk in index key order with few or no large gaps between them; -1 if statistics are not collected.
DENSITY	INTEGER		Y	Ratio of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index, expressed as a percent (integer between 0 and 100); -1 if statistics are not collected.
PAGE_FETCH_PAIRS[4]	VARCHAR (520)		Y	A list of pairs of integers, represented in character form. Each pair represents the number of pages in a hypothetical buffer, and the number of page fetches required to scan the table with this index using that hypothetical buffer. Zero-length string if no data is available.
NUMRIDS[4]	BIGINT		Y	Total number of row identifiers (RIDs) or block identifiers (BIDs) in the index; -1 if not known.
NUMRIDS_DELETED[4]	BIGINT		Y	Total number of row identifiers (or block identifiers) in the index that are marked deleted, excluding those identifiers on leaf pages on which all the identifiers are marked deleted.
NUM_EMPTY_LEAFS	BIGINT		Y	Total number of index leaf pages that have all of their row identifiers (or block identifiers) marked deleted.
AVERAGE_RANDOM_ FETCH_PAGES[1,2,4]	DOUBLE		Y	Average number of random table pages between sequential page accesses when fetching using the index; -1 if not known.
AVERAGE_RANDOM_ PAGES[2]	DOUBLE		Y	Average number of random table pages between sequential page accesses; -1 if not known.
AVERAGE_SEQUENCE_ GAP[2]	DOUBLE		Y	Gap between index page sequences. Detected through a scan of index leaf pages, each gap represents the average number of index pages that must be randomly fetched between sequences of index pages; -1 if not known.

Table 281. SYSSTAT.INDEXES Catalog View (continued)

Column Name	Data Type	Nullable	Updat-able	Description
AVERAGE_SEQUENCE_ FETCH_GAP[1,2,4]	DOUBLE		Y	Gap between table page sequences when fetching using the index. Detected through a scan of index leaf pages, each gap represents the average number of table pages that must be randomly fetched between sequences of table pages; -1 if not known.
AVERAGE_SEQUENCE_ PAGES[2]	DOUBLE		Y	Average number of index pages that are accessible in sequence (that is, the number of index pages that the prefetchers would detect as being in sequence); -1 if not known.
AVERAGE_SEQUENCE_ FETCH_PAGES[1,2,4]	DOUBLE		Y	Average number of table pages that are accessible in sequence (that is, the number of table pages that the prefetchers would detect as being in sequence) when fetching using the index; -1 if not known.
AVGPARTITION_ CLUSTERRATIO[3,4]	SMALLINT		Y	Degree of data clustering within a single data partition. -1 if the table is not partitioned, if statistics are not collected, or if detailed statistics are collected (in which case AVGPARTITION_ CLUSTERFACTOR will be used instead).
AVGPARTITION_ CLUSTERFACTOR[3,4]	DOUBLE		Y	Finer measurement of the degree of clustering within a single data partition. -1 if the table is not partitioned, if statistics are not collected, or if the index is defined on a nickname.
AVGPARTITION_PAGE_ FETCH_PAIRS[3,4]	VARCHAR (520)		Y	A list of paired integers in character form. Each pair represents a potential buffer pool size and the corresponding page fetches required to access a single data partition from the table. Zero-length string if no data is available, or if the table is not partitioned.
DATAPARTITION_ CLUSTERFACTOR	DOUBLE		Y	A statistic measuring the ″clustering″ of the index keys with regard to data partitions. It is a number between 0 and 1, with 1 representing perfect clustering and 0 representing no clustering.
INDCARD	BIGINT		Y	Cardinality of the index. This might be different from the cardinality of the table for indexes that do not have a one-to-one relationship between the table rows and the index entries.
PCTPAGESSAVED	SMALLINT			Approximate percentage of pages saved in the index as a result of index compression. -1 if statistics are not collected.
AVGLEAFKEYSIZE	INTEGER		Y	Average index key size for keys on leaf pages in the index.

Table 281. SYSSTAT.INDEXES Catalog View (continued)

Column Name	Data Type	Nullable	Updat-able	Description
AVGNLEAFKEYSIZE	INTEGER		Y	Average index key size for keys on non-leaf pages in the index.

Note:

1. When using DMS table spaces, this statistic cannot be computed.
2. Prefetch statistics are not gathered during a LOAD...STATISTICS YES, or a CREATE INDEX...COLLECT STATISTICS operation, or when the database configuration parameter *seqdetect* is turned off.
3. AVGPARTITION_CLUSTERRATIO, AVGPARTITION_CLUSTERFACTOR, and AVGPARTITION_PAGE_FETCH_PAIRS measure the degree of clustering within a single data partition (local clustering). CLUSTERRATIO, CLUSTERFACTOR, and PAGE_FETCH_PAIRS measure the degree of clustering in the entire table (global clustering). Global clustering and local clustering values can diverge significantly if the table partitioning key is not a prefix of the index key, or when the table partitioning key and the index key are logically independent of each other.
4. This statistic cannot be updated if the index type is 'XPTH' (an XML path index).
5. Because logical indexes on an XML column do not have statistics, the SYSSTAT.INDEXES catalog view excludes rows whose index type is 'XVIL'.

SYSSTAT.ROUTINES

Each row represents a user-defined routine (scalar function, table function, sourced function, method, or procedure). Does not include built-in functions.

Table 282. SYSSTAT.ROUTINES Catalog View

Column Name	Data Type	Nullable	Updat-able	Description
ROUTINESCHEMA	VARCHAR (128)			Schema name of the routine if ROUTINEMODULENAME is null; otherwise schema name of the module to which the routine belongs.
ROUTINEMODULENAME	VARCHAR (128)			Unqualified name of the module to which the routine belongs. The null value if not a module routine.
ROUTINENAME	VARCHAR (128)			Unqualified name of the routine.
ROUTINETYPE	CHAR (1)			Type of routine. • F = Function • M = Method • P = Procedure
SPECIFICNAME	VARCHAR (128)			Name of the routine instance (might be system-generated).
IOS_PER_INVOC	DOUBLE		Y	Estimated number of inputs/outputs (I/Os) per invocation; 0 is the default; -1 if not known.
INSTS_PER_INVOC	DOUBLE		Y	Estimated number of instructions per invocation; 450 is the default; -1 if not known.
IOS_PER_ARGBYTE	DOUBLE		Y	Estimated number of I/Os per input argument byte; 0 is the default; -1 if not known.

Table 282. SYSSTAT.ROUTINES Catalog View (continued)

Column Name	Data Type	Nullable	Updat-able	Description
INSTS_PER_ARGBYTE	DOUBLE		Y	Estimated number of instructions per input argument byte; 0 is the default; -1 if not known.
PERCENT_ARGBYTES	SMALLINT		Y	Estimated average percent of input argument bytes that the routine will actually read; 100 is the default; -1 if not known.
INITIAL_IOS	DOUBLE		Y	Estimated number of I/Os performed the first time that the routine is invoked; 0 is the default; -1 if not known.
INITIAL_INSTS	DOUBLE		Y	Estimated number of instructions executed the first time the routine is invoked; 0 is the default; -1 if not known.
CARDINALITY	BIGINT		Y	Predicted cardinality of a table function; -1 if not known, or if the routine is not a table function.
SELECTIVITY	DOUBLE		Y	For user-defined predicates; -1 if there are no user-defined predicates.

SYSSTAT.TABLES

Each row represents a table, view, alias, or nickname. Each table or view hierarchy has one additional row representing the hierarchy table or hierarchy view that implements the hierarchy. Catalog tables and views are included.

Table 283. SYSSTAT.TABLES Catalog View

Column Name	Data Type	Nullable	Updat-able	Description
TABSCHEMA	VARCHAR (128)			Schema name of the object.
TABNAME	VARCHAR (128)			Unqualified name of the object.
CARD	BIGINT		Y	Total number of rows in the table; -1 if statistics are not collected.
NPAGES	BIGINT		Y	Total number of pages on which the rows of the table exist; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
FPAGES	BIGINT		Y	Total number of pages; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
OVERFLOW	BIGINT		Y	Total number of overflow records in the table; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
CLUSTERED	CHAR (1)	Y		• Y = Table is multidimensionally clustered (even if only by one dimension) • Null value = Table is not multidimensionally clustered
ACTIVE_BLOCKS	BIGINT		Y	Total number of active blocks in the table, or -1. Applies to multidimensional clustering (MDC) tables only.

Table 283. SYSSTAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Updat-able	Description
AVGCOMPRESSEDROWSIZE	SMALLINT		Y	Average length (in bytes) of compressed rows in this table; -1 if statistics are not collected.
AVGROWCOMPRESSIONRATIO	REAL		Y	For compressed rows in the table, this is the average compression ratio by row; that is, the average uncompressed row length divided by the average compressed row length; -1 if statistics are not collected.
AVGROWSIZE	SMALLINT			Average length (in bytes) of both compressed and uncompressed rows in this table; -1 if statistics are not collected.
PCTROWSCOMPRESSED	REAL		Y	Compressed rows as a percentage of the total number of rows in the table; -1 if statistics are not collected.
PCTPAGESSAVED	SMALLINT		Y	Approximate percentage of pages saved in the table as a result of row compression. This value includes overhead bytes for each user data row in the table, but does not include the space that is consumed by dictionary overhead; -1 if statistics are not collected.

Database object topics

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 where the database manager will place your table space data. Then, the database manager manages the container and space allocation for the table spaces as you create and populate them.

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.

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 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.

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*.

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 prior to 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, respectively. 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.

Note: I/O access to long field (LF) data and large object (LOB) data will be buffered for both SMS and DMS containers, regardless of the setting for the table space in question.

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 non-DB2 files, 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 sortheap can be increased. In Version 9.5, DB2NTNOCACHE is deprecated and might be removed in a future release. 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.

Setting the current instance environment variables

About this task

When you run commands to start or stop an instance's database manager, DB2 applies the command to the current instance. DB2 determines the current instance as follows:

- If the **DB2INSTANCE** environment variable is set for the current session, its value is the current instance. To set the **DB2INSTANCE**, enter:

  ```
  set db2instance=<new_instance_name>
  ```

- If **DB2INSTANCE** is not set for the current session, the DB2 database manager uses the setting for the **DB2INSTANCE** environment variable from the system environment variables. On Windows, system environment variables are set in the System Environment registry.

- If **DB2INSTANCE** is not set at all, the DB2 database manager uses the registry variable, **DB2INSTDEF**.

  To set the **DB2INSTDEF** registry variable at the global level of the registry, enter:

  ```
  db2set db2instdef=<new_instance_name> -g
  ```

To determine which instance applies to the current session, enter:

```
db2 get instance
```

System environment variables

DB2_ALTERNATE_GROUP_LOOKUP

- Operating system: AIX
- Default: NULL, Values: NULL or GETGRSET
- 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.

DB2DBMSADDR

- Operating system: Linux on x86 and Linux on zSeries® (31-bit)
- Default: NULL, Values: virtual addresses in the range 0x09000000 to 0xB0000000 in increments of 0x10000
- The **DB2DBMSADDR** registry variable specifies the default database shared memory address in hexadecimal format.

  Note: An incorrect address can cause severe issues with the DB2 database system, ranging from an inability to start a DB2 instance, to an ability 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=
  ```

  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.

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 DB2 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.

  DB2 servers versions 7.1 or later support **DB2DOMAINLIST**, but only in a pure Windows domain environment. Starting with Version 8 FixPak

15 and Version 9.1 Fix Pack 3, **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 "Setting the current instance environment variables" on page 1063 and com.ibm.db2.luw.admin.regvars.doc/doc/t0004957.dita.

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. Beginning with version 9.5, 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 com.ibm.db2.luw.admin.regvars.doc/doc/t0004957.dita.

DB2OPTIONS

- Operating system: All
- Default: NULL
- Used to set the command line processor options.

DB2_PARALLEL_IO

- Operating system: All
- Default: NULL, 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 tablespace 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 284. 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.

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.

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 is displayed in the system level of the Control Center's object tree to aid administrators in the identification of server systems that can be administered from the Control Center.

  When using the Search the Network function of the Configuration Assistant, DB2 discovery returns this name and it is displayed at the system level in the resulting object tree. 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

- 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, updates and requests follow the version 9.5 behavior.
- Beginning with version 9.5, updates or changes to a database configuration act across all database partitions, when you do not specify a partition clause. **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.

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, 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.

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
- Design Advisor
- Control Center database information panel
- 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.

General registry variables

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.

– 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.

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.

DB2COUNTRY

- Operating system: Windows
- Default: NULL, Values: all valid numeric country, territory, or region codes
- This variable specifies the country, territory, or region code of the client application. When specified, this value overrides the operating system setting.

Note: DB2COUNTRY is deprecated and might be removed in a future release. Instead, use **DB2TERRITORY**, which accepts the same values as **DB2COUNTRY**

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.

DB2DBMSADDR

- Operating system: Windows 32-bit
- Default: 0x20000000, Values: 0x20000000 to 0xB0000000 in increments of 0x10000
- This variable specifies the default database manager shared memory address in hexadecimal format. If db2start fails due to a shared memory address collision, this registry variable can be modified to force the database manager instance to allocate its shared memory at a different address.

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_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.

DB2FFDC

- Operating system: All
- Default: ON, Values: ON, CORE:OFF
- This variable provides the ability to deactivate core file generation. By default, this registry variable is set to ON. If this registry variable is not set, or is set to a value other than CORE:OFF, core files may be generated if the DB2 server abends.

  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. 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.

On Linux operating systems, the default core file size limit is set to `0` (that is, `ulimit -c`). With this setting, core files are not generated. To allow core files to be created on Linux operating systems, set the value to unlimited.

Note: **DB2FFDC** is being deprecated in version 9.5, and will be removed in a later release. The new registry variable **DB2FODC** incorporates **DB2FFDC**'s functionality.

DB2FODC

- Operating system: All
- Default: The concatenation of all FODC parameters (see below)
  - for Linux and UNIX: "COMMAND=*val* DUMPCORE=ON DUMPDIR=*diagpath*"
  - for Windows: "DUMPCORE=ON 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 gigabytes, 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 override 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. One exception is AIX where an ulimit setting of "unlimited" will be overridden with a value of 8 GB for DB2 server processes only. If you require a core dump larger than 8 GB, set ulimit to an appropriately large value, such as the size of RAM, or set **CORELIMIT** with a sufficiently large value.

    Note: Any changes to the user core limit or **CORELIMIT** are not effective until the next recycling of the DB2 instance.

  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. This option may be used for other large binary dumps that have to be stored outside of FODC package, not for only core files.

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.

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 DB2 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_CONNRETRIESis 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_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
- 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
  ```

  Note: Currently, this registry variable only has settings for Linux; additional settings for other operating systems will be added in future releases.
  OLD_CPU_USAGE will be restored to its default setting.
- 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.

Administration configuration topics

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 statements
- SQL program variable data
- Output data from the server processing an SQL 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.

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 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.

Appendix B. Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Help for DB2 tools
  - 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 e-mail 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 e-mail 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/shop/publications/order. English and translated DB2 Version 9.7 manuals in PDF format can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg2700947.

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 285. DB2 technical information

Name	Form Number	Available in print	Last updated
Administrative API Reference	SC27-2435-00	Yes	August, 2009
Administrative Routines and Views	SC27-2436-00	No	August, 2009
Call Level Interface Guide and Reference, Volume 1	SC27-2437-00	Yes	August, 2009
Call Level Interface Guide and Reference, Volume 2	SC27-2438-00	Yes	August, 2009
Command Reference	SC27-2439-00	Yes	August, 2009
Data Movement Utilities Guide and Reference	SC27-2440-00	Yes	August, 2009
Data Recovery and High Availability Guide and Reference	SC27-2441-00	Yes	August, 2009
Database Administration Concepts and Configuration Reference	SC27-2442-00	Yes	August, 2009
Database Monitoring Guide and Reference	SC27-2458-00	Yes	August, 2009
Database Security Guide	SC27-2443-00	Yes	August, 2009
DB2 Text Search Guide	SC27-2459-00	Yes	August, 2009
Developing ADO.NET and OLE DB Applications	SC27-2444-00	Yes	August, 2009
Developing Embedded SQL Applications	SC27-2445-00	Yes	August, 2009
Developing Java Applications	SC27-2446-00	Yes	August, 2009
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-2447-00	No	August, 2009
Developing User-defined Routines (SQL and External)	SC27-2448-00	Yes	August, 2009
Getting Started with Database Application Development	GI11-9410-00	Yes	August, 2009
Getting Started with DB2 Installation and Administration on Linux and Windows	GI11-9411-00	Yes	August, 2009

Table 285. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
Globalization Guide	SC27-2449-00	Yes	August, 2009
Installing DB2 Servers	GC27-2455-00	Yes	August, 2009
Installing IBM Data Server Clients	GC27-2454-00	No	August, 2009
Message Reference Volume 1	SC27-2450-00	No	August, 2009
Message Reference Volume 2	SC27-2451-00	No	August, 2009
Net Search Extender Administration and User's Guide	SC27-2469-00	No	August, 2009
Partitioning and Clustering Guide	SC27-2453-00	Yes	August, 2009
pureXML Guide	SC27-2465-00	Yes	August, 2009
Query Patroller Administration and User's Guide	SC27-2467-00	No	August, 2009
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC27-2468-00	No	August, 2009
SQL Procedural Languages: Application Enablement and Support	SC27-2470-00	Yes	August, 2009
SQL Reference, Volume 1	SC27-2456-00	Yes	August, 2009
SQL Reference, Volume 2	SC27-2457-00	Yes	August, 2009
Troubleshooting and Tuning Database Performance	SC27-2461-00	Yes	August, 2009
Upgrading to DB2 Version 9.7	SC27-2452-00	Yes	August, 2009
Visual Explain Tutorial	SC27-2462-00	No	August, 2009
What's New for DB2 Version 9.7	SC27-2463-00	Yes	August, 2009
Workload Manager Guide and Reference	SC27-2464-00	Yes	August, 2009
XQuery Reference	SC27-2466-00	No	August, 2009

Table 286. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
Installing and Configuring DB2 Connect Personal Edition	SC27-2432-00	Yes	August, 2009
Installing and Configuring DB2 Connect Servers	SC27-2433-00	Yes	August, 2009

Table 286. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print	Last updated
DB2 Connect User's Guide	SC27-2434-00	Yes	August, 2009

Table 287. Information Integration technical information

Name	Form Number	Available in print	Last updated
Information Integration: Administration Guide for Federated Systems	SC19-1020-02	Yes	August, 2009
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-04	Yes	August, 2009
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-02	No	August, 2009
Information Integration: SQL Replication Guide and Reference	SC19-1030-02	Yes	August, 2009
Information Integration: Introduction to Replication and Event Publishing	GC19-1028-02	Yes	August, 2009

Ordering printed DB2 books

About this task

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* DVD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation DVD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation DVD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2luw/v9r7.

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at http://www.ibm.com/shop/publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.

- To order printed DB2 books from your local IBM representative:
  1. Locate the contact information for your local representative from one of the following Web sites:
     - The IBM directory of world wide contacts at www.ibm.com/planetwide
     - The IBM Publications Web site at http://www.ibm.com/shop/publications/order. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.
  2. When you call, specify that you want to order a DB2 publication.
  3. Provide your representative with the titles and form numbers of the books that you want to order. For titles and form numbers, see "DB2 technical library in hardcopy or PDF format" on page 1085.

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.

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

About this task

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/.

Displaying topics in your preferred language in the DB2 Information Center

About this task

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

- To display topics in your preferred language in the Internet Explorer browser:

1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
2. Ensure your preferred language is specified as the first entry in the list of languages.
   – To add a new language to the list, click the **Add...** button.

     Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
   – To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

- To display topics in your preferred language in a Firefox or Mozilla browser:
  1. Select the button in the **Languages** section of the **Tools** —> **Options** —> **Advanced** dialog. The Languages panel is displayed in the Preferences window.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
     – To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
     – To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

Results

On some browser and operating system combinations, you must also change the regional settings of your operating system to the locale and language of your choice.

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

Before you begin

A DB2 Version 9.7 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

About this task

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates - updates existing Information Center features and languages. An additional benefit of automatic updates is that the Information

Center is unavailable for a minimal period of time during the update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.

- Manual updates - should be used when you want to add features or languages during the update process. 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.

Procedure

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.

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/V9.7` directory.
   b. Navigate from the installation directory to the `doc/bin` directory.
   c. Run the `ic-update` script:
      ```
      ic-update
      ```
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 9.7` 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 `ic-update.bat` file:
      ```
      ic-update.bat
      ```

Results

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

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 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/db2icdv97 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 9.7 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/V9.7` 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 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 installation directory's `doc\bin` 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 installation directory's `doc/bin` 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/db2icdv97 start
     ```

Results

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 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/db2help/.

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.

"Visual Explain" in *Visual Explain Tutorial*
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

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 *DB2 Troubleshooting Guide* or the Database fundamentals section of the *DB2 Information Center*. There you will find information about how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 database products.

DB2 Technical Support Web site
Refer to the DB2 Technical Support Web site 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 DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/support/db2_9/

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial 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.

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.

Appendix C. 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 on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not 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.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 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 and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web

sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

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
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
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 of Sun Microsystems, Inc. in the United States, other countries, 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, Intel® Centrino®, Intel Centrino logo, Celeron®, Intel® Xeon®, 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

Z

Printed in USA

SC14-7213-00

Spine information:

IBM DB2 9.7 for Linux, UNIX, and Windows

Common Criteria Certification: Administration and User Documentation - Volume 1 - Revision 6